Red Hat Certificate System 8.1 Command-Line Tools Guide

a reference guide
Edition 1

Ella Deon Ballard

Red Hat Certificate System 8.1 Command-Line Tools Guide

a reference guide
Edition 1

Ella Deon Ballard
dlackey@redhat.com

Legal Notice

Copyright © 2012 Red Hat, Inc.

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This book covers important, Certificate System-specific, command-line tools that you can use to create, remove, and manage subsystem instances and to create and manage keys and certificates.

Table of Contents

About This Guide **5**
1. Required Concepts 5
2. What Is in This Guide 5
3. Common Tool Information 7
4. Examples and Formatting 7
4.1. Formatting for Examples and Commands 7
4.2. Tool Locations 7
4.3. Guide Formatting 7
5. Additional Reading 8
6. Giving Feedback 9
7. Document History 9

Chapter 1. pkicreate, pkisilent, and pkiremove (Create and Remove Instance Tools) . **11**
1.1. pkicreate 11
1.1.1. Syntax 11
1.1.2. Usage 13
1.2. pkisilent 14
1.2.1. Syntax 14
1.2.2. Usage 21
1.2.2.1. Configuring a CA 21
1.2.2.2. Configuring a DRM, OCSP, or TKS 22
1.2.2.3. Configuring an RA 23
1.2.2.4. Configuring a TPS 24
1.2.2.5. Cloning a Subsystem 25
1.2.2.6. Using Different Key Settings 26
1.2.2.7. Submitting Requests to an External CA 27
1.3. pkiremove 29
1.3.1. Syntax 30
1.3.2. Usage 30

Chapter 2. TokenInfo (Managing External Hardware Tokens) **31**
2.1. Syntax 31

Chapter 3. sslget (Downloading Files over HTTPS) **32**
3.1. Syntax 32
3.2. Usage 32

Chapter 4. AuditVerify (Audit Log Verification) **34**
4.1. Setting up the Auditor's Database 34
4.2. Syntax 35
4.3. Return Values 36
4.4. Usage 36
4.5. Results 36

Chapter 5. setpin (Generating Unique PINs for Entities) **38**
5.1. The setpin Command 38
5.1.1. Editing the setpin.conf Configuration File 38
5.1.2. Syntax 39
5.1.3. Usage 41
5.2. How setpin Works 41
5.2.1. Input File 43
5.2.2. Output File 44
5.2.3. How PINs Are Stored in the Directory 45

5.2.4. Exit Codes 45

Chapter 6. AtoB (Converting ASCII to Binary) 47
6.1. Syntax 47
6.2. Usage 47

Chapter 7. BtoA (COnverting Binary to ASCII) 48
7.1. Syntax 48
7.2. Usage 48

Chapter 8. PrettyPrintCert (Printing Certificates) 49
8.1. Syntax 49
8.2. Usage 49

Chapter 9. PrettyPrintCrl (Printing Readable CRLs)) 52
9.1. Syntax 52
9.2. Usage 52

Chapter 10. tkstool (Managing Token Keys) 54
10.1. Syntax 54
10.2. Usage 56

Chapter 11. CMCRequest (Creating CMC Requests) 60
11.1. Syntax 60
11.2. Usage 63
11.3. Output 64

Chapter 12. CMCEnroll (Performing CMC Enrollments) 72
12.1. Syntax 72
12.2. Usage 72
12.3. Output 73

Chapter 13. CMCResponse (Parsing a CMC Response) 76
13.1. Syntax 76
13.2. Usage and Output 76

Chapter 14. CMCRevoke (Signing a Revocation Request) 83
14.1. Syntax 83
14.2. Testing CMC Revocation 84
14.3. Output 84

Chapter 15. CRMFPopClient (Sending an Encoded CRMF Request) 86
15.1. Syntax 86
15.2. Usage 87
15.3. Output 88

Chapter 16. ExtJoiner (Adding Cerificate Extensions to Requests) 93
16.1. Syntax 93
16.2. Usage 93

Chapter 17. GenExtKeyUsage (Adding the Key Usage Extension to a Request) 95
17.1. Syntax 95

Chapter 18. GenIssuerAltNameExt (Adding the Issuer Name Extention to a Request) 96
18.1. Syntax 96
18.2. Usage 97

Chapter 19. SubjectAltNameExt (Adding the Subject Alternative Name Extension to a Request) 98

19.1. Syntax 98
19.2. Usage 99

Chapter 20. HttpClient (Sends a Request over HTTP) 100
20.1. Syntax 100

Chapter 21. OCSPClient (Sending an OCSP Request) 102
21.1. Syntax 102

Chapter 22. PKCS10Client (Generating a PKCS #10 Certificate Request) 103
22.1. Syntax 103

Chapter 23. PKCS12Export (Exports Certificates and Keys from a Database) 104
23.1. Syntax 104
23.2. Usage and Output 104

Chapter 24. revoker (Sending Revocation Requests) 105
24.1. Syntax 105
24.2. Output 106

Chapter 25. tpsclient (Debugging the TPS) 112
25.1. Syntax 114

Chapter 26. DRMTool (Rewrapping Private Keys) 117
26.1. Syntax 117
26.2. .cfg File 120
26.3. Examples 126
26.4. Usage 128

Index 132
A 132
B 132
C 132
D 133
E 133
G 133
P 133
R 134
S 134
T 134

About This Guide

The *Certificate System Command-Line Tools Guide* describes the command-line tools and utilities bundled with Red Hat Certificate System and provides information such as command syntax and usage examples to help use these tools.

This guide is intended for experienced system administrators who are planning to deploy the Certificate System. Certificate System agents should use the *Certificate System Agent's Guide* for information on how to perform agent tasks, such as handling certificate requests and revoking certificates.

1. Required Concepts

This guide assumes familiarity with the following concepts:

- Public-key cryptography and the Secure Sockets Layer (SSL) protocol
  - SSL cipher suites
  - The purpose of and major steps in the SSL handshake
- Intranet, extranet, Internet security, and the role of digital certificates in a secure enterprise, including the following topics:
  - Encryption and decryption
  - Public keys, private keys, and symmetric keys
  - Significance of key lengths
  - Digital signatures
  - Digital certificates
  - The role of digital certificates in a public-key infrastructure (PKI)
  - Certificate hierarchies

2. What Is in This Guide

This guide contains the following topics:

Table 1. List of Contents

Chapter 1, *pkicreate, pkisilent, and pkiremove (Create and Remove Instance Tools)*	Describes the tools used to create and remove subsystem instances.
Section 1.2, “pkisilent”	Describes the tool used for a silent instance creation.
Chapter 2, *TokenInfo (Managing External Hardware Tokens)*	Describes the utility which can be used to identify tokens on a machine, which shows whether the Certificate System can detect those tokens to use for a subsystem.
Chapter 3, *sslget (Downloading Files over HTTPS)*	Describes a tool used by the Certificate System to help configure and use security domains.
Chapter 4, *AuditVerify (Audit Log Verification)*	Describes how to use the tool used to verify signed audit logs.
Chapter 5, *setpin (Generating Unique PINs for Entities)*	Describes how to use the tool for generating unique PINs for end users and for populating their directory entries with PINs.
Chapter 6, *AtoB (Converting ASCII to Binary)*	Describes how to use the tool for converting ASCII data to its binary equivalent.
Chapter 7, *BtoA (COnverting Binary to ASCII)*	Describes how to use the tool for converting binary data to its ASCII equivalent.
Chapter 8, *PrettyPrintCert (Printing Certificates)*	Describes how to use the tool for printing or viewing the contents of a certificate stored as ASCII base-64 encoded data in a human-readable form.
Chapter 9, *PrettyPrintCrl (Printing Readable CRLs))*	Describes how to use the tool for printing or viewing the contents of a CRL stored as ASCII base-64 encoded data in a human-readable form.
Chapter 10, *tkstool (Managing Token Keys)*	Describes how to manipulate symmetric keys, including keys stored on tokens, the TKS master key, and related keys and databases.
Chapter 11, *CMCRequest (Creating CMC Requests)*	Describes how to construct a Certificate Management Messages over Cryptographic Message Syntax (CMC) request.
Chapter 12, *CMCEnroll (Performing CMC Enrollments)*	Describes how to sign a CMC certificate enrollment request with an agent's certificate.
Chapter 13, *CMCResponse (Parsing a CMC Response)*	Describes how to parse a CMC response.
Chapter 14, *CMCRevoke (Signing a Revocation Request)*	Describes how to sign a CMC revocation request with an agent's certificate.
Chapter 15, *CRMFPopClient (Sending an Encoded CRMF Request)*	Describes how to generate Certificate Request Message Format (CRMF) requests with proof of possession (POP).
Chapter 16, *ExtJoiner (Adding Cerificate Extensions to Requests)*	Describes how to use the tool for joining MIME-64 encoded formats of certificate extensions to create a single blob.
Chapter 17, *GenExtKeyUsage (Adding the Key Usage Extension to a Request)*	Describes how to generate a distinguished encoding rules (DER)-encoded Extended Key Usage extension.

Chapter 18, *GenIssuerAltNameExt (Adding the Issuer Name Extention to a Request)*	Describes how to generate an Issuer Alternative Name extension in base-64 encoding.
Chapter 19, *SubjectAltNameExt (Adding the Subject Alternative Name Extension to a Request)*	Describes how to generate a Subject Alternative Name extension in base-64 encoding.
Chapter 20, *HttpClient (Sends a Request over HTTP)*	Describes how to communicate with any HTTP/HTTPS server.
Chapter 21, *OCSPClient (Sending an OCSP Request)*	Describes how to verify certificate status by submitting Online Certificate Status Protocol (OCSP) requests to an instance of an OCSP subsystem.
Chapter 22, *PKCS10Client (Generating a PKCS #10 Certificate Request)*	Describes how to generate a Public-Key Cryptography Standards (PKCS) #10 enrollment request.
Chapter 24, *revoker (Sending Revocation Requests)*	Describes how to automate user management scripts to revoke certificates.
Chapter 25, *tpsclient (Debugging the TPS)*	Describes how to test the TPS configuration and common operations.

3. Common Tool Information

All of the tools in this guide are located in the **/usr/bin** directory, except for the Silent Install tool, which is downloaded separately and installed to any directory. These tools can be run from any location without specifying the tool location.

4. Examples and Formatting

4.1. Formatting for Examples and Commands

All of the examples for Red Hat Certificate System commands, file locations, and other usage are given for Red Hat Enterprise Linux 5.6 (32-bit) systems. Be certain to use the appropriate commands and files for your platform.

> **Example 1. Example Command**
>
> To start the Red Hat Certificate System:
>
> ```
> service pki-ca start
> ```

4.2. Tool Locations

All of the tools for Red Hat Certificate System are located in the **/usr/bin** directory. These tools can be run from any location without specifying the tool location.

4.3. Guide Formatting

Certain words are represented in different fonts, styles, and weights. Different character formatting is used to indicate the function or purpose of the phrase being highlighted.

Formatting Style	Purpose

Monospace font	Monospace is used for commands, package names, files and directory paths, and any text displayed in a prompt.
`Monospace with a background`	This type of formatting is used for anything entered or returned in a command prompt.
Italicized text	Any text which is italicized is a variable, such as *instance_name* or *hostname*. Occasionally, this is also used to emphasize a new term or other phrase.
Bolded text	Most phrases which are in bold are application names, such as **Cygwin**, or are fields or options in a user interface, such as a **User Name Here:** field or **Save** button.

Other formatting styles draw attention to important text.

NOTE

A note provides additional information that can help illustrate the behavior of the system or provide more detail for a specific issue.

IMPORTANT

Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.

WARNING

A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

5. Additional Reading

The documentation for Certificate System includes the following guides:

- *Certificate System Deployment Guide* describes basic PKI concepts and gives an overview of the planning process for setting up Certificate System.

  This manual is intended for Certificate System administrators.
- *Certificate System Installation Guide* covers the installation process for all Certificate System subsystems.

  This manual is intended for Certificate System administrators.
- *Certificate System Administrator's Guide* explains all administrative functions for the Certificate System. Administrators maintain the subsystems themselves, so this manual details backend configuration for certificate profiles, publishing, and issuing certificates and CRLs. It also covers managing subsystem settings like port numbers, users, and subsystem certificates.

  This manual is intended for Certificate System administrators.
- *Certificate System Agent's Guide* describes how agents — users responsible for processing

certificate requests and managing other aspects of certificate management — can use the Certificate System subsystems web services pages to process certificate requests, key recovery, OCSP requests and CRLs, and other functions.

This manual is intended for Certificate System agents.

- *Managing Smart Cards with the Enterprise Security Client* explains how to install, configure, and use the Enterprise Security Client, the user client application for managing smart cards, user certificates, and user keys.

  This manual is intended for Certificate System administrators, agents, privileged users (such as security officers), and regular end users.
- *Using End User Services* is a quick overview of the end-user services in Certificate System, a simple way for users to learn how to access Certificate System services.

  This manual is intended for regular end users.
- *Certificate System Command-Line Tools Guide* covers the command-line scripts supplied with Red Hat Certificate System.

  This manual is intended for Certificate System administrators.
- *Certificate System Migration Guide* covers version-specific procedures for migrating from older versions of Certificate System to Red Hat Certificate System 8.1.

  This manual is intended for Certificate System administrators.
- *Release Notes* contains important information on new features, fixed bugs, known issues and workarounds, and other important deployment information for Red Hat Certificate System 8.1.

All of the latest information about Red Hat Certificate System and both current and archived documentation is available at http://docs.redhat.com.

6. Giving Feedback

If there is any error in this *Command-Line Tools Guide* or there is any way to improve the documentation, please let us know. Bugs can be filed against the documentation for Red Hat Certificate System through Bugzilla, http://bugzilla.redhat.com/bugzilla. Make the bug report as specific as possible, so we can be more effective in correcting any issues:

- Select the Red Hat Certificate System product.
- Set the component to **`Doc - cli-tools-guide`**.
- Set the version number to 8.1.
- For errors, give the page number (for the PDF) or URL (for the HTML), and give a succinct description of the problem, such as incorrect procedure or typo.

  For enhancements, put in what information needs to be added and why.
- Give a clear title for the bug. For example, **`"Incorrect command example for setup script options"`** is better than **`"Bad example"`**.

We appreciate receiving any feedback — requests for new sections, corrections, improvements, enhancements, even new ways of delivering the documentation or new styles of docs. You are welcome to contact Red Hat Content Services directly at docs@redhat.com.

7. Document History

Revision 8.1.1-3	**December 20, 2013**	**Ella Deon Ballard**

Bug fixing.

Revision 8.1-15 **January 6, 2012** **Ella Deon Lackey**

Correcting hash and saltattribute descriptions and usage for the setpin tool, per Bugzilla #743012 and #747721.

Updating DRMTool usage and examples, per Bugzilla #749053.

Revision 8.1-7 **September 28, 2011** **Ella Deon Lackey**

Adding a note on ECC support, per Bugzilla #677416.

Revision 8.1-6 **July 25, 2011** **Ella Deon Lackey**

Editing the DRMTool information, Bugzilla #707075.

Fixing CRMFPop example command.

Revision 8.1-5 **July 7, 2011** **Ella Deon Lackey**

Adding DRMTool information.

Revision 8.1-4 **June 27, 2011** **Ella Deon Lackey**

Adding a draft for the new DRMTool.

Revision 8.1-3 **May 26, 2011** **Ella Deon Lackey**

Removed redundant **-a** description for AuditVerify syntax, Bugzilla #68199.

Adding required LDAP database parameters to cloning example, from Bugzilla #704622.

Revision 8.1-2 **May 20, 2011** **Ella Deon Lackey**

Fixed typo (trailing >) in the CRMFPopClient example.

Revision 8.1-1 **April 21, 2011** **Ella Deon Lackey**

Adding new pkiremove parameter, from Bugzilla #696175.

Revision 8.1-0 **March 21, 2011** **Ella Deon Lackey**

Initial draft for Certificate System 8.1 documentation for review.

Chapter 1. pkicreate, pkisilent, and pkiremove (Create and Remove Instance Tools)

The Certificate System includes three tools to create, configure, and remove subsystem instances: **pkicreate**, **pkisilent**, and **pkiremove**.

NOTE

The **pkicreate** tool does not install the Certificate System system; this is done through installing the packages or running the Red Hat Enterprise Linux **yum** command. This tool creates new instances after the default subsystems have been installed.
Likewise, the **pkiremove** utility does not uninstall the Certificate System subsystem packages; it removes a single subsystem instance.

1.1. pkicreate

The **pkicreate** tool creates instances of Certificate System subsystems and does a minimal configuration of the new instance, such as setting the configuration directory and port numbers. Further configuration is done through the HTML configuration pages.

1.1.1. Syntax

The **pkicreate** command has slightly different syntax depending on the type of subsystem being created because of port differences between the subsystem types. For a CA, OCSP, DRM, or TKS, the tool has the following syntax:

pkicreate -pki_instance_root=***/directory/path*** -subsystem_type=***type*** -pki-instance_name=***instance_ID*** [[-secure_port=***SSLport***] | [-agent_secure_port=***SSLport*** -ee_secure_port=***SSLport*** -ee_secure_client_auth_port=***SSLport_CA_only*** -admin_secure_port=***SSLport***]] -unsecure_port=***port*** -tomcat_server_port=***port*** [-user=***user_name***] [-group=***group_name***] [-redirect conf=***conf_directory***] [-redirect logs=***log_directory***] [-sans_security_manager] [-verbose] [-help]

For an RA or TPS, the tool has the following syntax:

pkicreate -pki_instance_root=***/directory/path*** -subsystem_type=***type*** -pki-instance_name=***instance_ID*** -secure_port=***SSLport*** -non_clientauth_secure_port=***SSLport*** -unsecure_port=***port*** [-user=***user_name***] [-group=***group_name***] [-audit_group=***group_name***] [-redirect conf=***conf_directory***] [-redirect logs=***log_directory***] [-verbose] [-help]

Parameter	Description
pki_instance_root	Gives the full path to the new instance configuration directory.
subsystem_type	Gives the type of subsystem being created. The possible values are as follows: - **ca**, for a Certificate Authority - **ra**, for a Registration Authority - **kra**, for a DRM - **ocsp**, for an OCSP - **tks**, for a TKS

	tps, for a TPS
pki_instance_name	Gives the name of the new instance. Instance names must be unique on a single machine, but do not have to be unique within the security domain (since instances are identified by hostname and port, not instance name).
secure_port	Sets a single SSL port number for the subsystem. This parameter is required if port separation is not configured, meaning that separate ports are not assigned for the administrator, agent, and end-entities services.
agent_secure_port	Sets the SSL port for the agent web services. If this is specified, then both **ee_secure_port** and **admin_secure_port** must be specified. For CAs only, an end-entities client authentication port is also required with the **ee_secure_client_auth_port** option.
ee_secure_port	Sets the SSL port for the end-entities web services. If this is specified, then both **agent_secure_port** and **admin_secure_port** must be specified. For CAs only, an end-entities client authentication port is also required with the **ee_secure_client_auth_port** option.
ee_secure_client_auth_port	*For CAs only.* Sets the SSL port for the end-entity client authentication. If this is specified, then **ee_secure_port**, **agent_secure_port**, and **admin_secure_port** must be specified.
admin_secure_port	Sets the SSL port number for the administrator services, usually the **pkiconsole**. If this is specified, then both **agent_secure_port** and **ee_secure_port** must be specified. For CAs only, an end-entities client authentication port is also required with the **ee_secure_client_auth_port** option.
non_clientauth_secure_port	Sets the end entities SSL port for RA and TPS subsystems.
unsecure_port	Sets the regular port number. If this is not set, the number is randomly generated. Still, it is recommended that administrators set this value to make sure there are no conflicts with SELinux labels for other services.
tomcat_server_port	Sets the port number for the Tomcat web server. This option must be set for CA, OCSP, TKS, and DRM instances. **tomcat_server_port** is not used when creating a TPS or RA instance since those subsystems do not use a Tomcat web server.
redirect conf	*Optional.* Sets the location for the configuration files for the new instance. This should include an

	instance-specific directory name in the path. For example, for the **pki-ca** instance, this should be something like **/etc/pki-ca**.
redirect logs	*Optional.* Sets the location for the log files for the new instance. This should include an instance-specific directory name in the path. For example, for the **pki-ca** instance, this should be something like **/var/log/pki-ca**.
user	*Optional.* Sets the user as which the Certificate System instance will run. This option must be set.
group	*Optional.* Sets the group as which the Certificate System instance will run. This option must be set.
audit_group	*Optional. For the TPS only.* Gives the name of the group for auditors for the TPS instance. The default is **pkiaudit**, if this option is not given.
sans_security_manager	*Optional. For the CA, OCSP, DRM, and TKS.* Configures the new instance to run without a Java Security Manager.
verbose	*Optional.* Runs the new instance creation in verbose mode.
help	Shows the help information.

1.1.2. Usage

In Example 1.1, “pkicreate Usage with a Single SSL Port”, the **pkicreate** is used to create a new CA instance running on ports **9080** and **9543**, named **pki-ca2**, in the **/var/lib/pki-*name*/pki-ca2** directory.

Example 1.1. pkicreate Usage with a Single SSL Port

```
pkicreate   -pki_instance_root=/var/lib   -subsystem_type=ca   -
pki_instance_name=pki-ca2   -secure_port=9543   -unsecure_port=9080   -
tomcat_server_port=1802   -user=pkiuser   -group=pkiuser   -verbose
```

Alternatively, the CA services can run on different ports. Example 1.2, “pkicreate with Port Separation” creates a CA instance with port separation. The agent port is **9544**, the end-entity port is **9543**, and the administrator port is **9545**.

Example 1.2. pkicreate with Port Separation

```
pkicreate   -pki_instance_root=/var/lib   -subsystem_type=ca   -
pki_instance_name=pki-ca2   -agent_secure_port=9544   -ee_secure_port=9543   -
ee_secure_client_auth_port=9546   -admin_secure_port=9545   -unsecure_port=9080
 -tomcat_server_port=1702   -user=pkiuser   -group=pkiuser   -verbose
```

While not required, it is a good idea to set the directories for the log and configuration files to keep the proper file structure.

Example 1.3. pkicreate with Specified Directory Locations

```
pkicreate -pki_instance_root=/var/lib -subsystem_type=ca -
pki_instance_name=pki-ca2 -agent_secure_port=9544 -ee_secure_port=9543 -
ee_secure_client_auth_port=9546 -admin_secure_port=9545 -unsecure_port=9080
 -tomcat_server_port=1702 -redirect conf /var/lib/pki-name/conf -redirect
log /var/log/pki-name/logs -user=pkiuser -group=pkiuser -verbose
```

1.2. pkisilent

The Certificate System includes a tool, **pkisilent**, which configures an instance through the command line, without having to access the HTML services pages.

NOTE

While **pkicreate** and **pkiremove** are installed with the other Red Hat Certificate System packages, the **pkisilent** tool must be downloaded independently. To install it, run **yum install pki-silent**.

Run **pkisilent** on a system which already has a subsystem installed, since this tool depends on having libraries, JRE, and core jar files already installed.

Two files are installed for the **pkisilent** tool:

- **pkisilent**, the Perl wrapper script.
- **pkisilent.jar**, the jar files containing the Java classes to perform a silent installation.

The utility can be downloaded and saved to any location and is then executed locally.

1.2.1. Syntax

The **pkisilent** script can be used to configure a new subsystem instance. This tool has the following syntax:

pkisilent Configure***type options***

It is also possible to configure a clone CA or DRM instance using **pkisilent**. This requires additional parameters to retrieve the master subsystem's keys.

pkisilent ... -clone true -clone_p12_file ***p12-file*** -clone_p12_password ***password*** -clone_start_tls true|false

TIP

There are two template files that are shell scripts for silent configuration: **/usr/share/pki/silent/pki_silent.template** and **/usr/share/pki/silent/subca_silent.template**. Both of these templates have detailed information on parameters and usage options for **pkisilent**.
To check the specific options for any **Configure***type* option, just run the **pkisilent** command with the **Configure***type* option and the **-help** flag. For example, to get the help for configuring a subordinate CA:

```
pkisilent ConfigureSubCA -help
```

The **Configure***type* option sets what kind of subsystem is being configured. This can be any of the following:

- ConfigureCA (for a root CA) or ConfigureSubCA (for a subordinate CA)
- ConfigureRA
- ConfigureDRM
- ConfigureOCSP
- ConfigureTKS
- ConfigureTPS

Table 1.1. Parameters for pkisilent

Parameter	Description
Basic Instance Configuration	
cs_hostname	The hostname for the Certificate System machine.
cs_port	The administrative SSL port number of the Certificate System instance.
subsystem_name	Sets the name of the new subsystem instance.
client_certdb_dir	The directory for the subsystem certificate databases.
client_certdb_pwd	The password to protect the certificate database.
preop_pin	The preoperation PIN number used for the initial configuration.
token_name	Gives the name of the HSM token used to store the subsystem certificates. This is only required for hardware tokens; if this parameter is not given, then the script automatically uses the local software token.
token_pwd	Gives the password for the HSM.
Agent and Admin User Configuration	
admin_user	The new admin user for the new subsystem.
admin_email	The email address of the admin user.
admin_password	The password for the admin user.
agent_key_size	The key size to use for generating the agent certificate and key pair.
agent_key_type	The key type to use for generating the agent certificate and key pair.
agent_cert_subject	The subject name for the agent certificate.
Security Domain Configuration	
domain_name	The name of the security domain to which the subsystem will be added.
sd_hostname	The hostname of the CA which hosts security domain.
sd_admin_port	The admin SSL port of the CA which hosts security domain.
sd_agent_port	The agent SSL port of the CA which hosts security domain.
sd_ssl_port	The end-entities SSL port of the CA which hosts security domain.
sd_admin_name	The username of the administrative user for the CA hosting the security domain.
sd_admin_password	The password of the administrative user for the CA hosting the security domain.
Internal Database Configuration	
ldap_host	The hostname of the Directory Server machine.
ldap_port	The non-SSL port of the Directory Server.
bind_dn	The bind DN which will access the Directory

	Server; this is normally the Directory Manager ID.
bind_password	The bind DN password.
base_dn	The entry DN under which to create all of the subsystem entries.
db_name	The database name.
secure_conn	Whether to use SSL to connect to the internal database. This is either true or false.
remove_data	Whether to overwrite the data if a database of the same name exsits.
Subsystem Certificates and Keys Configuration	
key_type	The type of key to generate; this can be **`rsa`** or **`ec`** [a].
key_size	The size of the key to generate. The default key size for RSA keys is 2048 bits; and for ECC keys, the default is 256 bits.
key_algorithm	The hashing algorithm to use for the key pair. The default is **`SHA256withRSA`** for RSA keys and **`SHA256withEC`** for ECC keys. RSA options: • SHA256withRSA • SHA1withRSA • SHA256withRSA • SHA512withRSA • MD5withRSA • MD2withRSA For ECC: • SHA256withEC (the default) • SHA1withEC • SHA384withEC • SHA512withEC
key_curvename	*For ECC keys*. The curve to use for the key. The default is **`nistp256`**.
signing_key_type signing_key_size signing_key_algorithm signing_key_curvename signing_key_signingalgorithm	*For CA signing certificates. CAs only*. Sets the specific settings to generate a CA signing key and certificate. The **`key_type`**, **`key_size`**, **`key_algorithm`**, and **`key_curvename`** parameters apply to every key and certificate generated by a susbsystem. However, each individual key can have its own parameters set separately. The parameters available to **`key_type`**, **`key_size`**, **`key_algorithm`**, and **`key_curvename`** apply to the CA signing key parameters.
ocsp_signing_key_type ocsp_signing_key_size	*For OCSP signing certificates. CAs and OCSPs*. Sets the specific settings to generate an OCSP signing key and certificate.

ocsp_signing_key_algorithm ocsp_signing_key_curvename ocsp_signing_key_signingalgorithm	The **key_type**, **key_size**, **key_algorithm**, and **key_curvename** parameters apply to every key and certificate generated by a susbsystem. However, each individual key can have its own parameters set separately. The parameters available to **key_type**, **key_size**, **key_algorithm**, and **key_curvename** apply to the OCSP signing key parameters.
audit_signing_key_type audit_signing_key_size audit_signing_key_algorithm audit_signing_key_curvename	*For audit signing certificates. For CA, DRM, OCSP, TKS, and TPS*. Sets the specific settings to generate an audit log signing key and certificate. The only supported key type for audit certificates is RSA. The **key_type**, **key_size**, **key_algorithm**, and **key_curvename** parameters apply to every key and certificate generated by a susbsystem. However, each individual key can have its own parameters set separately. The parameters available to **key_type**, **key_size**, **key_algorithm**, and **key_curvename** apply to the audit log signing key parameters.
subsystem_key_type subsystem_key_size subsystem_key_algorithm subsystem_key_curvename	*For subsystem client certificates. For all subsystems*. Sets the specific settings to generate an SSL client key and certificate. The **key_type**, **key_size**, **key_algorithm**, and **key_curvename** parameters apply to every key and certificate generated by a susbsystem. However, each individual key can have its own parameters set separately. The parameters available to **key_type**, **key_size**, **key_algorithm**, and **key_curvename** apply to the SSL client key parameters.
sslserver_key_type sslserver_key_size sslserver_key_algorithm sslserver_key_curvename	*For server certificates. For all subsystems*. Sets the specific settings to generate an SSL server key and certificate. The **key_type**, **key_size**, **key_algorithm**, and **key_curvename** parameters apply to every key and certificate generated by a susbsystem. However, each individual key can have its own parameters set separately. The parameters available to **key_type**, **key_size**, **key_algorithm**, and **key_curvename** apply to the SSL server key parameters.
save_p12	Sets whether to export the keys and certificate information to a backup PKCS #12 file. **true** backs up the information; **false** does not back

	up the information. *Only for the CA subsystem.*
backup_pwd	The password to protect the PKCS #12 backup file containing the subsystem keys and certificates. *Not for use with TPS installation.*
backup_fname	The file to which to export the the PKCS #12 backup file.
ca_subsystem_cert_subject_name ca_ocsp_cert_subject_name ca_server_cert_subject_name ca_sign_cert_subject_name ca_audit_signing_cert_subject_name	The subject names for the CA subsystem certificates.
ra_subsystem_cert_subject_name ra_server_cert_subject_name ra_subsystem_cert_nickname ra_server_cert_nickname	The subject names for the RA subsystem certificates.
ocsp_ocsp_cert_subject_name ocsp_server_cert_subject_name ocsp_subsystem_cert_subject_name ocsp_audit_signing_cert_subject_name	The subject names for the OCSP subsystem certificates.
drm_storage_cert_subject_name drm_transport_cert_subject_name drm_server_cert_subject_name drm_subsystem_cert_subject_name drm_audit_signing_cert_subject_name	The subject names for the DRM subsystem certificates.
tks_subsystem_cert_subject_name tks_server_cert_subject_name tks_audit_signing_cert_subject_name	The subject names for the TKS subsystem certificates.
tps_subsystem_cert_subject_name tps_server_cert_subject_name tps_subsystem_cert_nickname	The subject names for the TPS subsystem certificates.

tps_server_cert_nickname	
Required Subsystem Configuration	
ca_hostname	The hostname for the CA subsystem which will issue the certificates for a subordinate CA, RA, DRM, OCSP, TKS, or TPS subsystem.
ca_port	The non-SSL port number of the CA.
ca_ssl_port	The SSL end entities port number of the CA.
drm_hostname	The hostname for the DRM subsystem to use to archive keys. *For the TPS only.*
drm_ssl_port	The SSL agent port number of the DRM. *For the TPS only.*
tks_hostname	The hostname for the TKS subsystem to use to derive keys. *For the TPS only.*
tks_ssl_port	The SSL agent port number of the TKS. *For the TPS only.*
Authentication Database Configuration (TPS only)	
ldap_auth_host	Gives the hostname of the LDAP directory database to use for the TPS subsystem token database. *Only for the TPS subsystem.*
ldap_auth_port	Gives the port number of the LDAP directory database to use for the TPS subsystem token database. *Only for the TPS subsystem.*
ldap_auth_base_dn	Gives the base DN in the LDAP directory tree of the TPS token database under which to create token entries. *Only for the TPS subsystem.*
External CA for Issuing Certificates	
external	Sets whether to submit the subsystem certificates to the configured CA or to an external CA. The options are **`true`** or **`false`**.
ext_csr_file	The output file to which to write the generated certificate requests for the subsystem certificates. *Step one of the silent configuration process.*
ext_cert_file	The input file for the certificates issued by the external CA. *Step two of the silent configuration process.*
ext_ca_cert_chain_file	The input file for the CA certificate chain for the external CA issuing the certificate. *Step two of the silent configuration process.*
Cloning Configuration	
clone	Sets whether the new instance is a clone. Its possible values are **`true`** or **`false`**.
clone_p12_file	The filename of the PKCS#12 file for the backed-up keys for the original instance. This must be in the clone instance's **`/var/lib/pki-`*`name`*`/alias`** directory.
clone_p12_password	The password to access the PKCS#12 file.
clone_start_tls	Whether to use Start TLS with replication

	between the clones. This opens a secure connection over a standard port.

[a] Certificate System does not include a module natively to enable ECC, but it is possible to load and use a third-party PKCS #11 module with ECC-enabled. This is covered in the *Installation Guide*.

1.2.2. Usage

1.2.2.1. Configuring a CA

The options are slightly different between the subsystems; the simplest to configure is a CA with a new security domain.

This configured a CA, creates a new security domain, backs up its keys, and self-signs its certificates. Any spaces in the arguments must be escaped.

Example 1.4. Configuring a Root CA

```
pkisilent ConfigureCA -cs_hostname localhost
          -cs_port 9445
          -subsystem_name "pki-ca2"
          -client_certdb_dir /tmp/
          -client_certdb_pwd password
          -preop_pin sYY8er834FG9793fsef7et5
          -domain_name "testca"
          -admin_user admin
          -admin_email "admin@example.com"
          -admin_password secret
          -agent_name "jsmith"
          -agent_key_size 2048
          -agent_key_type rsa
          -agent_cert_subject "cn=ca\ agent\ cert"
          -ldap_host server
          -ldap_port 389
          -secure_conn false
          -remove_data true
          -bind_dn "cn=directory\ manager"
          -bind_password secret
          -base_dn "o=pki-ca2"
          -db_name "server.example.com-pki-ca2"
          -key_size 2048
          -key_type rsa
          -key_algorithm SHA256withRSA
          -backup_pwd password
          -backup_fname /export/backup.p12
          -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca\
domain"
          -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
          -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
          -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
          -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\
domain"
          -token_name "internal"
```

A subordinate CA — along with the DRM, OCSP, and TKS — is configured to join an existing security domain and to have its certificates signed by an existing Certificate System CA (by default; it is also possible to use an external CA). All of the parameters should be on a single line.

Example 1.5. Configuring a Subordinate CA

```
pkisilent ConfigureCA -cs_hostname localhost
          -cs_port 9445
          -subsystem_name "pki-ca2"
          -client_certdb_dir /tmp/
          -client_certdb_pwd password
          -preop_pin sYY8er834FG9793fsef7et5
          -sd_hostname "domain.example.com"
          -sd_admin_port 9445
          -sd_agent_port 9443
          -sd_ssl_port 9444
          -sd_admin_name admin
          -sd_admin_password secret
          -admin_user admin
          -admin_email "admin@example.com"
          -admin_password secret
          -agent_name "jsmith"
          -agent_key_size 2048
          -agent_key_type rsa
          -agent_cert_subject "cn=ca\ agent\ cert"
          -ldap_host server
          -ldap_port 389
          -secure_conn false
          -remove_data true
          -bind_dn "cn=directory\ manager"
          -bind_password secret
          -base_dn "o=pki-ca2"
          -db_name "server.example.com-pki-ca2"
          -key_size 2048
          -key_type rsa
          -save_p12 true
          -backup_pwd password
          -backup_fname /export/backup.p12
          -ca_hostname server.example.com
          -ca_port 9180
          -ca_ssl_port 9443
          -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca\
domain"
          -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
          -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
          -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
          -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\
domain"
          -token_name "internal"
```

1.2.2.2. Configuring a DRM, OCSP, or TKS

A DRM, TKS, and OCSP subsystem is largely the same as a subordinate CA, but without the **-save_p12** option. All of the parameters should be on a single line.

Example 1.6. Configuring a DRM

```
pkisilent ConfigureDRM -cs_hostname localhost
          -cs_port 9445
          -subsystem_name "pki-kra"
          -client_certdb_dir /tmp/
          -client_certdb_pwd password
          -preop_pin sYY8er834FG9793fsef7et5
          -domain_name "example\ domain"
          -sd_hostname "domain.example.com"
          -sd_admin_port 9445
          -sd_agent_port 9443
          -sd_ssl_port 9444
          -sd_admin_name admin
          -sd_admin_password secret
          -admin_user admin
          -admin_email "admin@example.com"
          -admin_password secret
          -agent_key_size 2048
          -agent_key_type rsa
          -agent_cert_subject "cn=drm\ agent\ cert"
          -agent_name "jsmith"
          -ldap_host server
          -ldap_port 389
          -secure_conn false
          -remove_data true
          -bind_dn "cn=directory\ manager"
          -bind_password secret
          -base_dn "o=pki-kra"
          -db_name "server.example.com-pki-kra"
          -key_size 2048
          -key_type rsa
          -backup_pwd password
          -ca_hostname server.example.com
          -ca_port 9180
          -ca_ssl_port 9443
          -drm_subsystem_cert_subject_name "cn=drm\ subsystem\ cert,o=example\
domain"
          -drm_transport_cert_subject_name "cn=drm\ transport\ cert,o=example\
domain"
          -drm_server_cert_subject_name "cn=drm\ client\ cert,o=example\
domain"
          -drm_storage_cert_subject_name "cn=drm\ storage\ cert,o=example\
domain"
          -drm_audit_signing_cert_subject_name "cn=drm\ audit\ signing\
cert,o=example\ domain"
          -token_name "internal"
```

1.2.2.3. Configuring an RA

The RA, unlike the other subsystems, does not use an LDAP database, so it does not specify the same database parameters as the other subsystems. In this example, the keys for the RA are not automatically backed up and there is no audit log signing certificate, since the RA is the only subsystem which does not support signed audit logs.

Example 1.7. Configuring an RA

```
pkisilent ConfigureRA -cs_hostname localhost
          -cs_port 9445
          -subsystem_name "pki-ra2"
          -client_certdb_dir /tmp/
          -client_certdb_pwd password
          -preop_pin sYY8er834FG9793fsef7et5
          -domain_name "example\ domain"
          -sd_hostname "domain.example.com"
          -sd_admin_port 9445
          -sd_agent_port 9443
          -sd_ssl_port 9444
          -sd_admin_name admin
          -sd_admin_password secret
          -admin_user admin
          -admin_email "admin@example.com"
          -admin_password secret
          -agent_name "jsmith"
          -agent_key_size 2048
          -agent_key_type rsa
          -agent_cert_subject "cn=ra\ agent\ cert"
          -ca_hostname server.example.com
          -ca_port 9180
          -ca_ssl_port 9443
          -key_size 2048
          -key_type rsa
          -ra_subsystem_cert_subject_name "cn=ra\ subsystem\ cert,o=testca\
domain"
          -ra_server_cert_subject_name "cn=ra\ client\ cert,o=testca\ domain"
          -token_name ="internal"
```

1.2.2.4. Configuring a TPS

A TPS requires the most parameters, since it depends on having a CA, DRM, and TKS configured and uses two LDAP databases, along with joining an existing security domain.

Example 1.8. Configuring a TPS

```
pkisilent ConfigureTPS -cs_hostname localhost
          -cs_port 9445
          -subsystem_name "pki-tps2"
          -client_certdb_dir /tmp/
          -client_certdb_pwd password
          -preop_pin sYY8er834FG9793fsef7et5
          -domain_name "example\ domain"
          -sd_hostname "domain.example.com"
          -sd_admin_port 9445
          -sd_agent_port 9443
          -sd_ssl_port 9444
          -sd_admin_name admin
          -sd_admin_password secret
          -admin_user admin
          -admin_email "admin@example.com"
          -admin_password secret
          -agent_name "jsmith"
          -agent_key_size 2048
          -agent_key_type rsa
          -agent_cert_subject "cn=tps\ agent\ cert"
          -ldap_host server
          -ldap_port 389
          -secure_conn false
          -remove_data true
          -bind_dn "cn=directory\ manager"
          -bind_password secret
          -base_dn "o=pki-tps2"
          -db_name "server.example.com-pki-tps2"
          -ca_hostname server.example.com
          -ca_port 9180
          -ca_ssl_port 9443
          -tks_hostname server.example.com
          -tks_ssl_port 13443
          -drm_hostname server.example.com
          -drm_ssl_port 10443
          -key_size 2048
          -key_type rsa
          -tps_subsystem_cert_subject_name "cn=tps\ subsystem\ cert,o=testca\
domain"
          -tps_server_cert_subject_name "cn=tps\ client\ cert,o=testca\ domain"
          -tps_audit_signing_cert_subject_name "cn=audit\ signing\
cert,o=testca\ domain"
          -ldap_auth_host auth.example.com
          -ldap_auth_port 389
          -ldap_auth_base_dn "ou=tps,ou=People,dc=example,dc=com"
          -token_name "internal"
```

1.2.2.5. Cloning a Subsystem

pkisilent can be used to configure a clone for a CA or DRM. A clone mirrors and uses much of the same configuration as its master, and the clone options either reflect the same configuration settings as the master or actually draw on the master's configuration:

- The keys for the master's certificates, stored in its PKCS #12 backup files (-clone_p12_file and -clone_p12_password). The **.p12** file must be in the **/var/lib/pki-*name*/alias** directory for the

clone instance.

- The same issuing CA for certificates.
- The same security domain, set in the -sd_* parameters.
- The same LDAP base DN and database name, set in the -ldap_* parameters (either the hostname or the port must be different, since the clone does require a separate Directory Server instance).

This clones an existing CA (all parameters should be on one line):

```
pkisilent ConfigureCA -cs_hostname localhost
          -cs_port 9445
          -subsystem_name "clone-ca2"
          -client_certdb_dir /tmp/
          -client_certdb_pwd password
          -preop_pin sYY8er834FG9793fsef7et5
          -doman_name "example\ domain"
          -sd_hostname "domain.example.com"
          -sd_admin_port 9445
          -sd_agent_port 9443
          -sd_ssl_port 9444
          -sd_admin_name admin
          -sd_admin_password secret
          -admin_user admin
          -admin_email "admin@example.com"
   -admin_password secret
          -agent_name jsmith
          -agent_key_size 2048
          -agent_key_type rsa
          -key_type rsa
          -key_size 2048
   -agent_cert_subject "'CN=jsmith,ou=clone-ca2,o=Example Domain'"
          -ldap_host ldap-server.example.com
          -ldap_port 389
          -bind_dn "'cn=Directory Manager'"
          -bind_password secret
          -base_dn "dc=ca.example.com-clone-ca2"
          -db_name "ca.example.com-clone-ca2"
          -clone true
          -clone_p12_file backup.p12
          -clone_p12_password secret
          -clone_start_tls false
          -master_instance_name pki-ca
          -ca_hostname server.example.com
          -ca_non_ssl_port 9180
          -ca_ssl_port 9443
          -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca\
domain"
          -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
          -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
          -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
          -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\
domain"
          -token_name "internal"
```

1.2.2.6. Using Different Key Settings

Generally, the key settings are applied to all keys generated for a subsystem.

```
... -key_size 2048 -key_type rsa -key_algorithm SHA256withRSA  ...
```

However, each individual key can have its own parameters set separately, meaning each key could be of a different type, of a different size, or use different algorithms or curves. As with the certificate subject names, the types of keys that are configured differ depending on the subsystem.

Every key can be given a unique setting, or only the specified keys can be given unique settings, while all other keys use the default.

For example, this sets different settings for every key for a CA:

```
pkisilent ConfigureCA -cs_hostname localhost
          -cs_port 9445
          -subsystem_name "pki-ca2"
          -client_certdb_dir /tmp/
          -client_certdb_pwd password
          -preop_pin sYY8er834FG9793fsef7et5
          -domain_name "testca"
   -signing_key_type ec
   -signing_key_size 256
   -signing_key_curvename nist256
   -signing_key_signingalgorithm SHA256withEC
   -ocsp_signing_key_type ec
   -ocsp_signing_key_size 256
   -ocsp_signing_key_curvename nist256
   -ocsp_signing_key_signingalgorithm SHA256withEC
   -audit_signing_key_type rsa
   -audit_signing_key_size 2048
   -audit_signing_key_algorithm SHA256withRSA
   -subsystem_key_type rsa
   -subsystem_key_size  2048
   -subsystem_key_algorithm SHA512withRSA
   -sslserver_key_type rsa
   -sslserver_key_size 2048
   -sslserver_key_algorithm SHA256withRSA
   ...
```

This only sets values for the CA signing key and uses the defaults for the other keys:

```
pkisilent ConfigureCA -cs_hostname localhost
          -cs_port 9445
          -subsystem_name "pki-ca2"
   -signing_key_type ec
   -signing_key_size 256
   -signing_key_curvename nist256
   -signing_key_signingalgorithm SHA256withEC
          -key_size 2048
          -key_type rsa
   -key_algorithm SHA256withRSA
...
```

1.2.2.7. Submitting Requests to an External CA

A CA outside of the security domain can be used to generate a subsystem's certificates. It is also possible to request and submit certificates issued by an external CA using **pkisilent**.

By default, the **pkisilent** command assumes that you will request a certificate from a CA within the

security domain, and this CA is identified in the **-ca_hostname** and other **ca_** options. This assumes that the **-external** option is false.

To submit the subsystem certificate requests to an external CA, explicitly set the **-external** option to true. The generated certificate requests are exported to a file, and then can be submitted to the external CA. Once they are issued, files which contain the subsystem certificates and the CA certificate chain for the issuing external CA can be passed using the **pkisilent** command. This is set in four parameters:

Submitting certificates to an external CA is a three-step process, two of them involving **pkisilent**:

1. In the first step, much of the preliminary information is configured for the instance.

   Along with the subsystem configuration settings, the subsystem's certificate requests are generated and written to the file specified in **-ext_csr_file**. These certificate requests must be submitted to the external CA.

   For example (in real life, these options should be on a single line):

```
pkisilent ConfigureCA
        -cs_hostname server.example.com
        -cs_port 9445
        -subsystem_name "pki-ca2"
        -client_certdb_dir /tmp/
        -client_certdb_pwd password
        -preop_pin sYY8er834FG9793fsef7et5
        -domain_name "testca"
        -admin_user admin
        -admin_password secret
        -admin_email "admin@example.com"
        -agent_name jsmith
        -agent_key_size 2048
        -agent_key_type rsa
        -agent_cert_subject "cn=ca\ agent\ cert"
        -ldap_host ldapserver.example.com
        -ldap_port 389
        -secure_conn false
        -remove_data true
        -bind_dn "cn=directory\ manager"
        -bind_password password -base_dn "o=pki-ca2"
        -db_name "server.example.com-pki-ca2"
        -key_size 2048
        -key_type rsa
        -key_algorithm SHA512withRSA
        -token_name internal
        -token_pwd 242986083911
        -save_p12 true
        -backup_pwd password
        -backup_fname /export/backup.p12
        -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca\
domain"
        -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
        -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
        -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
        -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\
domain"
        -external true
        -ext_csr_file /tmp/cert.req
```

2. The certificate requests are submitted to the external CA, and the issued certificates are retrieved and saved to file.

3. The newly issued subsystem certificates are installed in the instance by referencing the saved certificate file in the **-ext_ca_cert_file** parameter and the issuing CA's certificate chain in the **-ext_ca_cert_chain_file** parameter.

   For example (in real life, these options should be on a single line):

```
pkisilent ConfigureCA
       -cs_hostname server.example.com
       -cs_port 9445
       -subsystem_name "pki-ca2"
       -client_certdb_dir /tmp/
       -client_certdb_pwd password
       -preop_pin sYY8er834FG9793fsef7et5
       -domain_name "testca"
       -admin_user admin
       -admin_password secret
       -admin_email "admin@example.com"
       -agent_name jsmith
       -agent_key_size 2048
       -agent_key_type rsa
       -agent_cert_subject "cn=ca\ agent\ cert"
       -ldap_host ldapserver.example.com
       -ldap_port 389
       -secure_conn false
       -remove_data true
       -bind_dn "cn=directory\ manager"
       -bind_password password
       -base_dn "o=pki-ca2"
       -db_name "server.example.com-pki-ca2"
       -key_size 2048
       -key_type rsa
       -key_algorithm SHA512withRSA
       -token_name internal
       -token_pwd 242986083911
       -save_p12 true
       -backup_pwd password
       -backup_fname /export/backup.p12
       -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca\
domain"
       -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
       -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
       -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
       -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\
domain"
       -external true
       -ext_ca_cert_file /tmp/cert.cer
       -ext_ca_cert_chain_file /tmp/cachain.cer
```

   This is also when the final configuration to create the administrator user is performed.

NOTE

All of the previous parameters must be included the second time that **pkisilent** is invoked.

1.3. pkiremove

The **pkiremove** tool removes subsystem instances. This tool removes the single subsystem instance specified; it does not uninstall the Certificate System packages.

1.3.1. Syntax

pkiremove -pki_instance_root=***/directory/path*** -pki_instance_name=***instance_ID*** [-token_pwd=***password***] [-force]

Parameter	Description
pki_instance_root	Gives the full path to the instance configuration directory.
pki_instance_name	Gives the name of the instance.
token_pwd	Specifies the password to use to access the NSS databases for the instance being removed. The remove script first looks for the password as a command argument. If the password is not given, then it checks the **password.conf** file. If the password is not in the file, then the command prompts for the password.
force	Causes the removal to proceed without confirmation.

1.3.2. Usage

The following example removes a DRM instance named **pki-drm2** which was installed in the **/var/lib/pki-*name*/pki-drm2** directory. Since no password is given, the script assumes the password is in **password.conf**.

```
pkiremove -pki_instance_root=/var/lib -pki_instance_name=pki-drm2 -force
```

Chapter 2. TokenInfo (Managing External Hardware Tokens)

This tool is used to determine which external hardware tokens are visible to the Certificate System subsystem. This can be used to diagnose whether problems using tokens are related to the Certificate System being unable to detect it.

2.1. Syntax

The **`TokenInfo`** tool has the following syntax:

```
TokenInfo /directory/alias
```

Option	Description
/directory/alias	Specifies the path and file to the certificate and key database directory; for example, **`/var/lib/pki-ca/alias`**.

Chapter 3. sslget (Downloading Files over HTTPS)

This tool is similar to the **wget** command, which downloads files over HTTP. **sslget** supports client authentication using NSS libraries. The configuration wizard uses this utility to retrieve security domain information from the CA.

3.1. Syntax

The **sslget** tool has the following syntax:

sslget [-e ***profile information***] -n ***rsa_nickname*** [[-p ***password***] | [-w ***passwordFile***]] [-d ***dbdir***] [-v] [-V] -r ***url hostname*** [:***port***]

Option	Description
e	*Optional.* Submits information through a subsystem form by specifying the form name and the form fields. For example, this can be used to submit certificate enrollments through a certificate profile.
n	Gives the CA certificate nickname.
p	Gives the certificate database password. Not used if the **-w** option is used.
w	*Optional.* Gives the password file path and name. Not used if the **-p** option is used.
d	*Optional.* Gives the path to the security databases.
v	*Optional.* Sets the operation in verbose mode.
V	*Optional.* Gives the version of the **sslget** tool.
r *url*	Gives the URL of the site or server from which to download the information. Depending on how DNS and the network are configured, this can be a machine name, fully-qualified domain name, or IPv4 or IPv6 address.
hostname	Gives the hostname of the server to which to send the request. Depending on how DNS and the network are configured, this can be a machine name, fully-qualified domain name, or IPv4 or IPv6 address.
port	*Optional.* Gives the port number of the server.

3.2. Usage

It is possible to use **sslget** to submit information securely to Certificate System subsystems. For example, to submit a certificate request through a certificate profile enrollment for to a CA, the command is as follows:

```
sslget -e "profileId=caInternalAuthServerCert&cert_request_type=pkcs10
&requestor_name=TPS-server.example.com-7889
&cert_request=MIIBGTCBxAIBADBfMSgwJgYDVQQKEx8yMDA2MTEwNngxMi
BTZmJheSBSZWRoYXQgRG9tYWluMRIwEAYDVQQLEwlyaHBraS10cHMxHzAdBgNVBA
MTFndhdGVyLnNmYmF5LnJlZGhhdC5jb20wXDANBgkqhkiG9w0BAQEFAANLADBIAk
EAsMcYjKD2cDJOeKjhuAiyaC0YVh8hUzfcrf7ZJlVyROQx1pQrHiHmBQbcCdQxNz
YK7rxWiR62BPDR4dHtQzj8RwIDAQABoAAwDQYJKoZIhvcNAQEEBQADQQAKpuTYGP
%2BI1k50tjn6enPV6j%2B2lFFjrYNwlYWBe4qYhm3WoA0tIuplNLpzP0vw6ttIMZ
kpE8rcfAeMG10doUpp
&xmlOutput=true&sessionID=-4771521138734965265
&auth_hostname=server.example.com&auth_port=9444"
 -d "/var/lib/pki-tps/alias" -p "password123" -v -n "Server-Cert cert-pki-tps" -r
"/ca/ee/ca/profileSubmit" server.example.com:9444
```

Chapter 4. AuditVerify (Audit Log Verification)

The **AuditVerify** tool is used to verify that signed audit logs were signed with the private signing key and that the audit logs have not been compromised.

Auditors can verify the authenticity of signed audit logs using the **AuditVerify** tool. This tool uses the public key of the signed audit log signing certificate to verify the digital signatures embedded in a signed audit log file. The tool response indicates either that the signed audit log was successfully verified or that the signed audit log was not successfully verified. An unsuccessful verification warns the auditor that the signature failed to verify, indicating the log file may have been tampered with (compromised).

4.1. Setting up the Auditor's Database

AuditVerify needs access to a set of security databases (usually the auditor's personal security databases) containing the signed audit log signing certificate and its chain of issuing certificates. One of the CA certificates in the issuance chain must be marked as trusted in the database.

Auditors should import the audit signing certificate into their personal certificate and key databases before running **AuditVerify**. The auditor should not use the security databases of the Certificate System instance that generated the signed audit log files. If there are no readily accessible certificate and key database, the auditor must create a set of certificate and key databases and import the signed audit log signing certificate chain.

NOTE

The **signedAudit** directory kept by the subsystem is not writeable by any user, including auditors.

IMPORTANT

The auditor user must be a member of the **pkiaudit** group, or whatever system group was identified as the audit group by the **-audit_group** parameter when **pkicreate** was run to create the subsystem.

To create the security databases and import the certificate chain:

1. Create a special directory in the auditor's home directory to use to perform the verification. For example:

```
mkdir ~jsmith/auditVerifyDir
```

2. Use the **certutil** tool to create an empty set of certificate databases in the auditor's home directory.

```
certutil -d ~jsmith/auditVerifyDir -N
```

3. Download the CA certificate from the CA's **Retrieval** page.

```
https://server.example.com:9444/ca/ee/ca/
```

4. Import the CA certificate and log signing certificate into the databases, marking the CA certificate as trusted. The certificates can be obtained from the CA in ASCII format.

   If the CA certificate is in a file called **cacert.txt** and the log signing certificate is in a file called **logsigncert.txt**, then the **certutil** is used to set the trust for the new audit security database directory pointing to those files, as follows:

```
certutil -d ~jsmith/auditVerifyDir/ -A -n "CA Certificate" -t "CT,CT,CT" -a -
i /var/lib/instance_ID/alias/cacert.txt

certutil -d ~jsmith/auditVerifyDir -A -n "Log Signing Certificate"-t ",,P" -a
-i /var/lib/instance_ID/alias/logsigncert.txt
```

4.2. Syntax

The **AuditVerify** tool has the following syntax:

```
AuditVerify -d dbdir -n signing_certificate_nickname -a logListFile [-P
cert/key_db_prefix] [-v]
```

Option	Description
a	Specifies the text file containing a comma separated list (in chronological order) of the signed audit logs to be verified. The contents of the *logListFile* are the full paths to the audit logs. For example: `/var/log/pki-ca/signedAudit/ca_cert-ca_audit, /var/log/pki-ca/signedAudit/ca_cert-ca_audit.20030227102711, /var/log/pki-ca/signedAudit/ca_cert-ca_audit.20030226094015` This file should be created in a directory which is writeable by the auditor, such as a special auditing directory like **~jsmith/auditDir**.
d	Specifies the directory containing the security databases with the imported audit log signing certificate. This directory is almost always the auditor's own personal certificate databases in a personal directory, such as **~jsmith/auditVerifyDir/**.
n	Gives the nickname of the certificate used to sign the log files. The nickname is whatever was used when the log signing certificate was imported into that database.
P	*Optional.* The prefix to prepend to the certificate and key database filenames. If used, a value of empty quotation marks (“”) should be specified for this argument, since the auditor is using separate certificate and key databases from the Certificate

	System instance and it is unlikely that the prefix should be prepended to the new audit security database files.
v	*Optional.* Specifies verbose output.

4.3. Return Values

When **AuditVerify** is used, one of the following codes is returned:

Return Value	Description
0	Indicates that the signed audit log has been successfully verified.
1	Indicates that there was an error while the tool was running.
2	Indicates that one or more invalid signatures were found in the specified file, meaning that at least one of the log files could not be verified.

4.4. Usage

After a separate audit database directory has been configured, do the following:

1. Create a text file containing a comma-separated list of the log files to be verified. The name of this file is referenced in the **AuditVerify** command.

   For example, this file could be **logListFile** in the **/etc/audit** directory. The contents are the comma-separated list of audit logs to be verified, such as "**auditlog.1213**, **auditlog.1214**, **auditlog.1215**."
2. If the audit databases do not contain prefixes and are located in the user home directory, such as **/home/smith/.mozilla**, and the signing certificate nickname is **"auditsigningcert"**, the **AuditVerify** command is run as follows:

```
AuditVerify -d ~jsmith/auitVerifyDir -n auditsigningcert -a
/etc/audit/logListFile -P "" -v
```

4.5. Results

The input file, **audit_list**, is a simple text file which gives the full path to the sifgned audit logs to be verified.

```
cat ~jsmith/auditVerifyDir/audit_list
 /var/lib/pki-ca/logs/signedAudit/ca_audit.20110211145833
```

If no modifications have been made to any of the files, then **AuditVerify** returns a message that all signatures are valid.

```
AuditVerify -d ~jsmith/auditVerifyDir -n "Log Signing Certificate" -a
~jsmith/auditVerifyDir/audit_list

Verification process complete.
Valid signatures: 20
Invalid signatures: 0
```

If there is a modification to a log file, then the signature is invalidated. In that case, **`AuditVerify`** says that there is an invalid signature and returns the name of the edited log file and the line number of the modification.

```
AuditVerify -d ~jsmith/auditVerifyDir -n "Log Signing Certificate" -a
~jsmith/auditVerifyDir/audit_list
======
File:
/var/lib/pki-ca/logs/signedAudit/ca_audit.20110211145833
======
Line 52: VERIFICATION FAILED: signature of /var/lib/pki-
ca/logs/signedAudit/ca_audit.20101213141439:48 to /var/lib/pki-
ca/logs/signedAudit/ca_audit.20101213141439:51

Verification process complete.
Valid signatures: 19
Invalid signatures: 1
```

Chapter 5. setpin (Generating Unique PINs for Entities)

For the Certificate System to use the **UidPwdPinDirAuth** authentication plug-in module, the authentication directory must contain unique PINs for each end entity which will be issued a certificate. The Certificate System provides a tool, the *PIN Generator*, which generates unique PINs for end-entity entries in an LDAP directory. The tool stores these PINs as hashed values in the same directory against the corresponding user entries. It also copies the PINs to a text file so that the PINs can be sent to the end entities.

5.1. The setpin Command

This chapter describes the syntax and arguments of the **setpin** tool and the expected responses. For information on generating and storing PINs in the user authentication directory, see the *Certificate System Administrator's Guide*.

5.1.1. Editing the setpin.conf Configuration File

The **setpin** tool can use a configuration file, **setpin.conf**, to store some of its required options. Before running **setpin**, modify this file to reflect the directory information, and set the **setpin** tool to use this file by doing the following:

1. Open the **setpin.conf** file.

```
cd /usr/lib/pki/native-tools
vi setpin.conf
```

2. Edit the directory parameters in the file to match the directory installation information.

```
#------- Enter the hostname of the LDAP server
host=localhost

#------- Enter the port number of the LDAP server
port=389

#------- Enter the DN of the Directory Manager user
binddn=CN=Directory Manager

#------- Enter the password for the Directory manager user
bindpw=

#    Enter the DN and password for the new pin manager user
pinmanager=cn=pinmanager,dc=example,dc=com
pinmanagerpwd=

#    Enter the base over which this user has the power
#    to remove pins
basedn=ou=people,dc=example,dc=com

## This line switches setpin into setup mode.
## Please do not change it.
setup=yes
```

3. Run **setpin**, and set the option file to **setpin.conf**.

```
setpin optfile=/usr/lib/pki/native-tools/setpin.conf
```

5.1.2. Syntax

The **setpin** has the following syntax:

setpin host=***host_name*** [port=***port_number***] binddn=***user_id*** [bindpw=***bind_password***] filter="***LDAP_search_filter***" [basedn=***LDAP_base_DN***] [[length=***PIN_length***] | [minlength=***minimum_PIN_length***] | [maxlength=***maximum_PIN_length***]] [gen=***character_type***] [case=upperonly] [hash=***algorithm***] [saltattribute=***LDAP_attribute_to_use_for_salt_creation***] [input=***file_name***] [output=***file_name***] [write] [clobber] [testpingen=***count***] [debug] [optfile=***file_name***] [setup [pinmanager=***pinmanager_user***] [pinmanagerpwd=***pinmanager_password***]]

Option	Description
host	*Required*. Specifies the LDAP directory to which to connect. Depending on how DNS and the network are configured, this can be a machine name, fully-qualified domain name, or IPv4 or IPv6 address.
port	Specifies the LDAP directory port to which to bind. The default port number is the default LDAP port, **389**.
binddn	*Required.* Specifies the user as whom the PIN Generator binds to the LDAP directory. This user account must have read/write access to the directory.
bindpw	Gives the password for the user ID set in the **binddn** option. If the bind password is not given at the command line, the tool prompts for it.
filter	*Required.* Sets the search filter for those DNs in the directory for which the tool should generate PINs.
basedn	Specifies the base DN under which to search for DNs. If this argument is not specified, the filter searches from the root.
length	Specifies the exact number a PIN must contain; the default is 6. Do not use with **minlength** or **maxlength**.
minlength	Sets the minimum length of the generated PINs. If used with **maxlength**, this sets the lower end of the range of the PIN length. Do not use with **length**.
maxlength	Sets the maximum length of the generated PINs. If used with **minlength**, this sets the upper end of the range of the PIN length. Do not use with **length**.
gen	Specifies the character type for PINs. The characters in the password can be constructed out of alphabetic characters (**RNG-alpha**), alphanumeric characters (**RNG-alphanum**), or any printable ASCII characters (**printableascii**).
case	Restricts the character cases to uppercase only;

	otherwise, the case is mixed. Restricting alphabetic characters to uppercase reduces the overall combinations for the password space significantly. Use **case** with **gen**.
hash	Specifies the message digest algorithm with which to hash the PINs before storing them in the authentication directory. **NOTE** This should be set to **none** (which does not hash PINs) because the Directory Server may have restrictions on incoming hashed passwords. The default is **sha1**, which produces a 160-bit message digest. **md5** produces a 128-bit message digest. **none** does not hash the PINs.
saltattribute	Specifies the LDAP attribute to use for salt creation. This must be set to ***dn***. If an attribute is set, the tool integrates the value of the attribute with each PIN and hashes the resulting string with the hash routine. For details, refer to Section 5.2.3, “How PINs Are Stored in the Directory”. This attribute is ignored if the ***hash*** value is set to **none**, which is the recommended setting.
input	Specifies the file that contains the list of DNs to process. If this is used, the tool compares the filtered DNs to the ones in the input file and generates PINs for only those DNs .
output	Specifies the absolute path to the file to write the PINs as **setpin** generates them. If a file is not set, then the output is written to the standard output. Regardless of whether an output file is set, all error messages are directed to the standard error.
write	Sets whether the tool should write PINs to the directory. If specified, the PINs are written to the directory as they are generated. Otherwise, the tool does not make any changes to the directory. Do not write PINs to the directory if the PINs are to be checked. The PINs can be viewed in the output file to make sure that they are being assigned to the correct users and that they conform to the length and character restrictions. For more information, see Section 5.2.2, “Output File”.
clobber	Overwrites pre-existing PINs, if any, associated

	with a DN. If this option is not used, any existing PINs are left in the directory.
testpingen	Tests the PIN-generation mode. *count* sets the total number of PINs to generate for testing.
debug	Writes debugging information to the standard error. If **debug=attrs** is specified, the tool writes more detailed information about each entry in the directory.
optfile	Sets the tool to read options, one per line, from a file. This allows all arguments to be put in a file, instead of typing them at the command line. One configuration file, **setpin.conf**, is located in the **/usr/lib/pki/native-tools** directory.
setup	Switches to setup mode, which allows the tool to add to the directory schema.
pinmanager	Specifies the PIN manager user that has permission to remove the PIN for the **basedn** specified. Used with the **setup** option.
pinmanagerpwd	Gives the password for the PIN manager user. Used with the **setup** option.

5.1.3. Usage

First, run the **setpin** command with its **optfile** option pointing to the **setpin.conf** file.

```
setpin optfile=/usr/lib/pki/native-tools/setpin.conf
```

The tool modifies the schema with a new attribute (by default, **pin**) and a new object class (by default, **pinPerson**), creates a **pinmanager** user, and sets the ACI to allow only the **pinmanager** user to modify the **pin** attribute.

Then, disable setup mode for the **setpin** command. Either comment out the **setup** line or change the value to no.

```
vim /usr/lib/pki/native-tools/setpin.conf

setup=no
```

After the setup is complete, then **setpin** can be used to generate PINs.

The following command generates PINs for all entries that have the **CN** attribute in their distinguished name in an LDAP directory named **csldap** listening on port **389**. The PIN Generator binds to the directory as **Directory Manager** and starts searching the directory from the base DN **dn=dc=example,dc=com** in the directory tree. Any existing PINs are overwritten with the new ones.

```
setpin host=csldap port=389 binddn="CN=directory manager" bindpw=password
filter="(cn=*)" basedn="dc=example,dc=com" clobber write hash=none
```

5.2. How setpin Works

The PIN Generator generates PINs for user entries in an LDAP directory and updates the directory with

these PINs. To run the **setpin** command, the following five options are required:

- The host name (**host**) and port number (**port**) of the LDAP server
- The bind DN (**binddn**) and password (**bindpw**)
- An LDAP filter (**filter**) for filtering out the user entries that require PINs

The **setpin** command looks like the following:

```
setpin host=csldap port=19000 binddn="CN=Directory Manager" bindpw=secret
filter="(ou=employees)" basedn="dc=example,dc=com"
```

This example queries the directory for all the entries in the **employees** organizational unit (**ou**). For each entry matching the filter, information is printed out to standard error and to the standard output.

Note

Because the PIN Generator makes a lot of changes to the directory, it is important to use the correct filter, or the wrong entries are modified. Using the **write** option is a safeguard because no changes are made to the directory unless that option is used. This allows the PINs to be verified before any entries are modified.

The information can be written to a different output file by using the **output** option; see Section 5.2.2, "Output File" for more information. The entries returned by the LDAP search filter can be further restricted by using an ASCII input file which lists the entry DNs; only entries matching those in the file are updated. The input file is set with the **input** option. The input file is not a substitute for the LDAP directory entries; the filter attribute must still be provided. For more information about the input file, refer to Section 5.2.1, "Input File". Figure 5.1, "Using an Input and Output File When Generating PINs" shows how the input and output files work with the **setpin** tool.

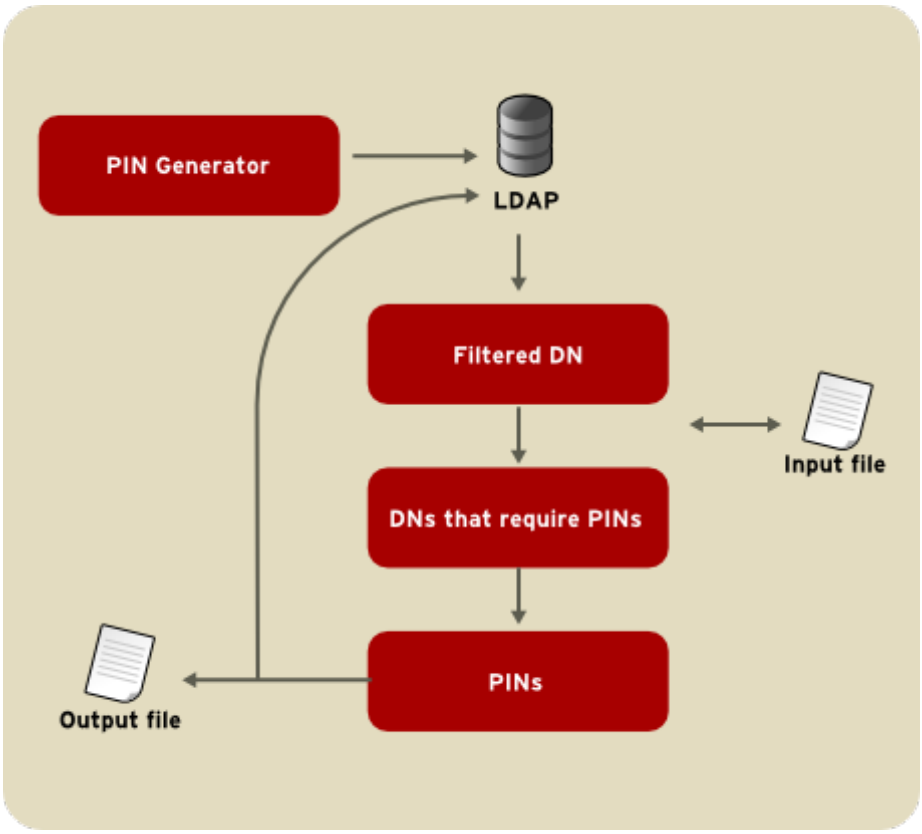

Figure 5.1. Using an Input and Output File When Generating PINs

The output file contains the entry and PIN information from running **setpin**, as shown in the following example:

```
Processing: cn=QA Managers,ou=employees,dc=example,dc=com
Adding new pin/password
dn:cn=QA Managers,ou=employees,dc=example,dc=com
pin:lDWynV
status:notwritten

Processing: cn=PD Managers,ou=employees,dc=example,dc=com
Adding new pin/password
dn:cn=PD Managers,ou=employees,dc=example,dc=com
pin:G69uV7
status:notwritten
```

The output also contains the status of each entry in the directory. The status values are listed in Table 5.1, “PIN Generator Status ”.

Table 5.1. PIN Generator Status

Exit Code	Description
notwritten	The PINs were not written to the directory because the **write** option was not used.
writefailed	The tool tried to modify the directory, but the write operation was unsuccessful.
added	The tool added the new PIN to the directory successfully.
replaced	The tool replaced an old PIN with a new one; this means the **clobber** option was used.
notreplaced	The tool did not replace the old PIN with a new one; this means the **clobber** option was not used.

If a PIN already exists for a user, it is not changed if the **setpin** command is run a second time. This allows new PINs to be created for new users without overwriting PINs for users who have already received a PIN. To overwrite a PIN, use the **clobber** option.

After making sure that the filter is matching the right users, run the **setpin** command again with the **write** option and with **output** set to the name of the file to capture the unhoused PINs. For details about the output file, refer to Section 5.2.2, “Output File”.

5.2.1. Input File

The PIN Generator can receive a list of DNs to modify in a text file specified by the **input** argument. If an input file is specified, then the tool compares the DNs returned by the filtered to the ones in the input file and updates only those DNs that match in the input file.

The input enables the user to provide the PIN Generator with an exact list of DNs to modify; it is also

possible to provide the PIN Generator with PINs in plain text for all DNs or for specific DNs.

There are two common situations when using an input file is useful:

- If PINs have been set for all entries in the user directory, and new users join the organization. For the new users to get certificates, the directory must contain PINs. PINs should be generated for only those two entries without changing any of the other user entries. Instead of constructing a complex LDAP filter, using an input file allows using a general filter, and the modified entries are restricted to the DNs of the two users listed in the input file.
- If a particular values, such as Social Security numbers, should be used as PINs, then the Social Security numbers can be put in the input file and provide those numbers as PINs to the PIN Generator. These are then stored as hashed values in the directory.

The format of the input file is the same as that of the output file (refer to Section 5.2.2, “Output File”) except for the status line. In the input file, PINs can be set for all the DNs in the file, for specific DNs, or for none of the DNs. If the PIN attribute is missing for a DN, the tool automatically generates a random PIN.

An input file looks like the following example:

```
dn:cn=user1, dc=example,dc=com

dn:cn=user2, dc=example,dc=com

...
dn:cn=user3, dc=example,dc=com
```

PINs can also be provided for the DNs in plain-text format; these PINs are hashed according to the command-line arguments.

```
dn:cn=user1, dc=example,dc=com
pin:pl229Ab

dn:cn=user2, dc=example,dc=com
pin:9j65dSf

...
dn:cn=user3, dc=example,dc=com
pin:3knAg60
```

NOTE

Hashed PINs cannot be provided to the tool.

5.2.2. Output File

The PIN Generator can capture the output to a text file specified by the **`output`** option.

The output contains a sequence of records in the following format:

```
dn: user_dn1
pin: generated_pin1
status: status1

dn: user_dn2
pin: generated_pin2
status: status2

...
dn: user_dn#
pin: generated_pin#
status: status#
```

where *user_dn* is a distinguished name matching the DN filter or listed in the input file. By default, the delimiter is a semi-colon (;) or the character defined on the command line. *generated_pin* is a string of characters of fixed or variable length, depending on the length parameters used. *status* is one of the values listed Table 5.1, “PIN Generator Status ”.

The first line in each record is always the DN. The subsequent lines for **pin** and **status** are optional. The record ends with a blank line, using the Unix end of line sequence (**\n**).

5.2.3. How PINs Are Stored in the Directory

Each PIN is concatenated with the corresponding LDAP attribute named in the **saltattribute** argument (which defaults to the entry DN, the recommended value). If this argument is not specified, the DN is used.

That string is hashed with the routine specified in the hash argument; the default algorithm is SHA-1, but this should be set to none so that it works with Directory Servers which restrict the use of incoming hashed passwords.

One byte is prepended to indicate the hash type used. The PIN is stored as follows:

```
byte[0] = X
```

The value of *X* depends on the hash algorithm chosen during the PIN generation process.

X	Hash Algorithm
0	SHA-1
1	MD5
45	none

The PIN is stored in the directory as a binary value, not as a base-64 encoded value.

5.2.4. Exit Codes

When the PIN Generator is finished running, it returns a result code showing how it ended. These result codes are listed in Table 5.2, “Result Codes Returned by the PIN Generator”.

Table 5.2. Result Codes Returned by the PIN Generator

Result Code	Description
0	The PIN generation was successful; PINs were set for all the DNs in the specified directory.
4	The tool could not bind to the directory as the user specified in the **`binddn`** parameter.
5	The tool could not open the output file specified in the **`output`** parameter.
7	There was an error parsing command-line arguments.
8	The tool could not open the input file specified in the **`input`** parameter.
9	The tool encountered an internal error.
10	The tool found a duplicate entry in the input file.
11	The tool did not find the salt attribute specified in the **`saltattribute`** parameter in the directory.

Chapter 6. AtoB (Converting ASCII to Binary)

The Certificate System ASCII to binary tool converts ASCII base-64 encoded data to binary base-64 encoded data.

6.1. Syntax

The ASCII to binary tool, **AtoB**, has the following syntax:

```
AtoB input_file output_file
```

Option	Description
input_file	Specifies the path and file to the base-64 encoded ASCII data.
output_file	Specifies the file where the utility should write the binary output.

6.2. Usage

The example command takes the base-64 ASCII data in the **ascii_data.in** file and writes the binary equivalent of the data to the **binary_data.out** file.

```
AtoB /usr/home/smith/test/ascii_data.in /usr/home/smith/test/binary_data.out
```

Chapter 7. BtoA (COnverting Binary to ASCII)

The Certificate System binary to ASCII tool, **BtoA** converts binary base-64 encoded data to ASCII base-64 encoded data.

7.1. Syntax

The **BtoA** tool uses the following syntax:

```
BtoA input_file output_file
```

Option	Description
input_file	Specifies the path and file of the base-64 encoded binary data.
output_file	Specifies the path and file to which the tool should write the ASCII output.

7.2. Usage

The following example of the **BtoA** utility takes the base-64 encoded binary data in the **binary_data.in** file and writes the ASCII equivalent of the data to the **ascii_data.out** file.

```
BtoA /usr/home/smith/test/binary_data.in /usr/home/smith/test/ascii_data.out
```

Chapter 8. PrettyPrintCert (Printing Certificates)

The Pretty Print Certificate utility, **PrettyPrintCert**, prints the contents of a certificate stored as ASCII base-64 encoded data to a readable format.

8.1. Syntax

The **PrettyPrintCert** command has the following syntax:

```
PrettyPrintCert [-simpleinfo] input_file [output_file]
```

Option	Description
simpleinfo	*Optional.* Prints limited certificate information in an easy to parse format.
input_file	Specifies the path to the file containing the ASCII base-64 encoded certificate.
output_file	*Optional.* Specifies the path and file to which the tool should write the certificate. If this option is not specified, the certificate information is written to the standard output.

8.2. Usage

The following example converts the ASCII base-64 encoded certificate in the **ascii_cert.in** file and writes the certificate in the pretty-print form to the output file **ascii_cert.out**.

```
PrettyPrintCert /usr/home/smith/test/ascii_cert.in
/usr/home/smith/test/ascii_cert.out
```

The base-64 encoded certificate data in the **ascii_cert.in** looks like the following:

```
-----BEGIN CERTIFICATE-----
MIIC2DCCAkGgAwIBAgICEAwwDQYJKoZIhvcNAQEFBQAwfDELMAkGA1UEBhMCVVMxIzA
hBgNVBAoTGlBhbG9va2FWaWxsZSBXaWRnZXRzLCBJbmMuMR0wGwYDVQQLExRXaWRnZX
QgTWFrZXJzICdSJyBVczEpMCcGA1UEAxMgVGVzdCBUZXN0IFRlc3QgVGVzdCBUZXN0I
FRlc3QgQ0EwHhcNOTkwMjE4MDMMzM5WhcNMDAwMjE4MDM0MzM5WjCBrjELMAkGA1UEB
hMCVVMxJjAkBgNVBAoTHU5ldHNjYXBlIENvbW11bmljYXRpb25zIENvcnAuMRUwEwYD
VQQLEwOZXRzY2FwZSBDTVMxGDAWBEBEwhtaGFybXNlbjEfMB0GA1UEAxWaW50ZGV2Y2
EgQWRtaW5pcwp0frfJOObeiSsia3BuifRHBNw95ZZQR9NIXr1x5bE
-----END CERTIFICATE-----
```

The certificate in pretty-print format in the **ascii_cert.out** file looks like the following:

```
Certificate:
Data:
Version: v3
Serial Number: 0x100C
Signature Algorithm: OID.1.2.840.113549.1.1.5 -1.2.840.113549.1.1.5
Issuer: CN=Test CA,OU=Widget Makers 'R'Us,O=Example Corporation,
Widgets\,Inc.,C=US
Validity:
 Not Before: Wednesday, February 17, 1999 7:43:39 PM
 Not After: Thursday, February 17, 2000 7:43:39 PM
Subject: MAIL=admin@example.com,CN=testCA Administrator, UID=admin, OU=IS,
 O=Example Corporation,C=US
Subject Public Key Info:
 Algorithm: RSA - 1.2.840.113549.1.1.1
 Public Key:
   30:81:89:02:81:81:00:DE:26:B3:C2:9D:3F:7F:FA:DF:
   24:E3:9B:7A:24:AC:89:AD:C1:BA:27:D1:1C:13:70:F7:
   96:59:41:1F:4D:21:7A:F5:C7:96:C4:75:83:35:9F:49:
   E4:B0:A7:5F:95:C4:09:EA:67:00:EF:BD:7C:39:92:11:
   31:F2:CA:C9:16:87:B9:AD:B8:39:69:18:CE:29:81:5F:
   F3:4D:97:B9:DF:B7:60:B3:00:03:16:8E:C1:F8:17:6E:
   7A:D2:00:0F:7D:9B:A2:69:35:18:70:1C:7C:AE:12:2F:
   0B:0F:EC:69:CD:57:6F:85:F3:3E:9D:43:64:EF:0D:5F:
   EF:40:FF:A6:68:FD:DD:02:03:01:00:01:
Extensions:
 Identifier: 2.16.840.1.113730.1.1
 Critical: no
 Value: 03:02:00:A0:
Identifier: Authority Key Identifier - 2.5.29.35
 Critical: no
 Key Identifier:
   EB:B5:11:8F:00:9A:1A:A6:6E:52:94:A9:74:BC:65:CF:
 07:89:2A:23:
Signature:
 Algorithm: OID.1.2.840.113549.1.1.5 - 1.2.840.113549.1.1.5
 Signature:
   3E:8A:A9:9B:D1:71:EE:37:0D:1F:A0:C1:00:17:53:26:
   6F:EE:28:15:20:74:F6:C5:4F:B4:E7:95:3C:A2:6A:74:
   92:3C:07:A8:39:12:1B:7E:C4:C7:AE:79:C8:D8:FF:1F:
   D5:48:D8:2E:DD:87:88:69:D5:3A:06:CA:CA:9C:9A:55:
   DA:A9:E8:BF:36:BC:68:6D:1F:2B:1C:26:62:7C:75:27:
   E2:8D:24:4A:14:9C:92:C6:F0:7A:05:A1:52:D7:CC:7D:
   E0:9D:6C:D8:97:3A:9C:12:8C:25:48:7F:51:59:BE:3C:
   2B:30:BF:EB:0A:45:7D:A6:49:FB:E7:BE:04:05:D6:8F:
```

The following example command takes the ASCII base-64 encoded certificate in the **`ascii_cert.in`** file and writes the information contained within the certificate to the simple format output file **`cert.simple`**.

```
PrettyPrintCert -simpleinfo /usr/home/smith/test/ascii_cert.in
/usr/home/smith/test/cert.simple
```

The base-64 encoded certificate data in **`ascii_cert.in`** file looks similar to the following:

```
-----BEGIN CERTIFICATE-----
MIIC2DCCAkGgAwIBAgICEAwwDQYJKoZIhvcNAQEFBQAwfDELMAkGA1UEBhMCVVMxIzA
hBgNVBAoTGlBhbG9va2FWaWxsZSBXaWRnZXRzLCBJbmMuMR0wGwYDVQQLExRXaWRnZX
QgTWFrZXJzICdSJyBVczEpMCcGA1UEAxMgVGVzdCBUZXN0IFRlc3QgVGVzdCBUZXN0I
FRlc3QgQ0EwHhcNOTkwMjE4MDMMzM5WhcNMDAwMjE4MDM0MzM5WjCBrjELMAkGA1UEB
hMCVVMxJjAkBgNVBAoTHU5ldHNjYXBlIENvbW11bmljYXRpb25zIENvcnAuMRUwEwYD
VQQLEwOZXRzY2FwZSBDTVMxGDAWBEBEwhtaGFybXNlbjEfMB0GA1UEAxWaW50ZGV2Y2
EgQWRtaW5pcwp0frfJOObeiSsia3BuifRHBNw95ZZQR9NIXr1x5bE
-----END CERTIFICATE-----
```

The simple certificate information in the **`cert.simple`** output file looks like the following:

```
  MAIL=admin@example.com
  CN=testCA Administrator
  UID=admin
  OU=IS
  O=Example Corporation
  C=US
```

Chapter 9. PrettyPrintCrl (Printing Readable CRLs))

The Pretty Print CRL tool, **PrettyPrintCrl**, prints the contents of a certificate revocation list (CRL) in an ASCII base-64 encoded file in a readable form.

9.1. Syntax

The **PrettyPrintCrl** utility has the following syntax:

```
PrettyPrintCrl input_file [output-file]
```

Option	Description
input_file	Specifies the path to the file that contains the ASCII base-64 encoded CRL.
output_file	*Optional.* Specifies the path to the file to write the CRL. If the output file is not specified, the CRL information is written to the standard output.

9.2. Usage

The following example **PrettyPrintCrl** command takes the ASCII base-64 encoded CRL in the **ascii_crl.in** file and writes the CRL in the pretty-print form to the output file **ascii_crl.out**.

```
PrettyPrintCrl /usr/home/smith/test/ascii_crl.in /usr/home/smith/test/ascii_crl.out
```

The base-64 encoded CRL in the **ascii_crl.in** file looks like the following:

```
-----BEGIN CRL-----
MIIBkjCBAIBATANBgkqhkiG9w0BAQQFADAsMREwDwYDVQQKEwhOZXRzY2FwZTEXMBUG
A1UEAxMOQ2VydDQwIFRlc3QgQ0EXDTk4MTIxNzIyMzcyNFowgaowIAIBExcNOTgxMjE
1MTMxODMyWjAMMAoGA1UdFQQDCgEBMCACARIXDTk4MTINTEzMjA0MlowDDAKBgNVHRU
EAwoBAjAgAgERFw05ODEyMTYxMjUxNTRaMAwwCgYDVR0VBAMKAQEwIAIBEBcNOTgxMj
E3MTAzNzI0WjAMMAoGA1UdFQQDCgEDMCACAQoXDTk4MTEyNTEzMTExOFowDDAKBgNVH
RUEAwoBATANBgkqhkiG9w0BQQFAAOBgQBCN85O0GPTnHfImYPROvoorx7HyFz2ZsuKs
VblTcemsX0NL7DtOa+MyY0pPrkXgm157JrkxEJ7GBOeogbAS6iFbmeSqPHj8+
-----END CRL-----
```

The CRL in pretty-print format in the **ascii_crl.out** output file looks like the following:

```
Certificate Revocation List:
Data:
Version: v2
Signature Algorithm: MD5withRSA - 1.2.840.113549.1.1.4
Issuer: CN=Test CA,O=Example Corporation
This Update: Thu Dec 17 14:37:24 PST 1998
Revoked Certificates:
Serial Number: 0x13
 Revocation Date: Tuesday, December 15, 1998 5:18:32 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: Key_Compromise
Serial Number: 0x12
 Revocation Date: Tuesday, December 15, 1998 5:20:42 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: CA_Compromise
Serial Number: 0x11
 Revocation Date: Wednesday, December 16, 1998 4:51:54 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: Key_Compromise
Serial Number: 0x10
 Revocation Date: Thursday, December 17, 1998 2:37:24 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: Affiliation_Changed
Serial Number: 0xA
 Revocation Date: Wednesday, November 25, 1998 5:11:18 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: Key_Compromise
Signature:
 Algorithm: MD5withRSA - 1.2.840.113549.1.1.4
 Signature:
   42:37:CE:4E:D0:63:D3:9C:77:C8:99:83:D1:3A:FA:28:
   AF:1E:C7:C8:5C:F6:66:CB:8A:B1:56:E5:4D:C7:A6:B1:
   7D:0D:2F:B0:ED:39:AF:8C:C9:8D:29:3E:B9:17:82:6D:
   79:EC:9A:E4:C4:42:7B:18:13:9E:A2:06:C0:4B:A8:85:
   6E:67:92:A8:F1:E3:F3:E2:41:1F:9B:2D:24:D9:DF:4C:
   2B:A1:68:CE:96:C7:AF:F7:5B:F7:3D:2F:06:57:39:74:
   CF:B2:FA:46:C6:AD:18:60:8D:3E:0C:F7:C1:66:52:37:
   CF:89:42:B0:D7:33:C4:95:7E:F4:D9:1E:32:B8:5E:12:
```

Chapter 10. tkstool (Managing Token Keys)

The TKS utility, **tkstool**, manages keys, including keys stored on tokens, the TKS master key, and related keys and databases.

10.1. Syntax

The **tkstool** can be used to manage certificates and keys in several different ways. The syntax for these different operations is as follows:

- Deleting a key from a token.

```
tkstool -D -n keyname -d dbdir [-h token_name] [-p dbprefix] [-f pwfile]
```

- Inputting shares to generate a new transport key.

```
tkstool -I -n keyname -d dbdir [-h token_name] [-p dbprefix] [-f pwfile]
```

- Displaying the key check value (KCV) of the specified key.

```
tkstool -K -n keyname -d dbdir [-h token_name] [-p dbprefix] [-f pwfile]
```

- Listing a specified key or all keys.

```
tkstool -L -n keyname -d dbdir [-h all | -h token_name]
 [-p dbprefix] [-f pwfile] [-x]
```

- Generating a new master key.

```
tkstool -M -n keyname -d dbdir [-h token_name] [-p dbprefix] [-f pwfile]
```

- Creating a new key database.

```
tkstool -N -d dbdir [-p dbprefix] [-f pwfile]
```

- Changing the key database password.

```
tkstool -P -d dbdir [-p dbprefix] [-f pwfile]
```

- Renaming a symmetric key.

```
tkstool -R -n keyname -r new_keyname -d dbdir [-h token_name]
 [-p dbprefix] [-f pwfile]
```

- Listing all security modules.

```
tkstool -S -d dbdir [-p dbprefix] [-x]
```

- Generating a new transport key.

```
tkstool -T -n keyname -d dbdir [-h token_name]
 [-p dbprefix] [-f pwfile] [-z noiseFile]
```

- Unwrapping a wrapped master key.

```
tkstool -U -n keyname -d dbdir -t transport_keyname -i inputFile
 [-h token_name] [-p dbprefix] [-f pwfile]
```

- Wrapping a new master key.

```
tkstool -W -n keyname -d dbdir -t transport_keyname -o outputFile
 [-h token_name] [-p dbprefix] [-f pwfile]
```

NOTE

Chrysalis-ITS version 2.3 is required to support version 1.0 of the **-R** option of the **tkstool**. Transport keys residing on Chrysalis-ITS hardware tokens created by an earlier version of **tkstool** cannot have their KCV values determined with the **-K** option of the **tkstool** because the **CKA_ENCRYPT** and **CKF_ENCRYPT** bits were not set when they were created by the previous tool.

The **tkstool** options are as follows:

Option	Description
D	Deletes a key from the token.
d	*Required.* Gives the security module database (HSM, if allowed for that operation) or the key database directory (software).
f	Gives the path and filename of the password file, if one is used.
h	Gives the token name for the toke which contains the key to be managed. Some operations allow an **all** option to manage all keys in the token.
I	Inputs shares to generate a new transport key.
i	*Required with -U.* Gives the path and filename of the input file which contains the wrapped master key.
K	Displays the KCV of the specified key.
L	Lists the specified key or all keys.
M	Generates a new master key.
N	Creates a new key database (software).
n	*Required for every operation except -N, -P, and -S.* Gives the name of the key being managed.
o	*Required with -W.* Gives the path and filename for the file to which to output the new wrapped master key.
P	Changes the key database password (software).
p	Gives the prefix to the key database directory.
R	Renames a symmetric key.
r	*Required with -R*. Gives the new key name.
S	Lists all security modules.

T	Generates a new transport key.
t	*Required with -U and -W.* Gives the name of the transport key being managed.
U	Unwraps the wrapped master key.
W	Wraps the new master key.
x	Forces the database to be read/write.
z	Gives the path and filename of the noise file to generate the key.

There are two additional options which can be used with **tkstool** to get more information about the utility.

Option	Description
H	Displays the extended help information.
V	Display the version number of the **tkstool** tool.

10.2. Usage

1. Check the version of **tkstool** by running the following command:

```
tkstool -V
```

This should return output similar to the following:

```
tkstool: Version 1.0
```

2. Create new software databases.

```
tkstool -N -d .
Enter a password which will be used to encrypt your keys.
The password should be at least 8 characters long,
and should contain at least one non-alphabetic character.

Enter new password:
Re-enter password:
```

NOTE

A hardware HSM can be used instead of the software database if the **modutil** utility is first used to insert the HSM slot and token into the **secmod.db** database.
If an HSM is used, then the option **-h** *hsm_token* must be added to each of commands below.

3. List the contents of the local software key database.

```
tkstool -L -d .

slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
tkstool: the specified token is empty
```

4. Create a transport key called **`transport`**.

```
tkstool -T -d . -n transport
```

5. When prompted, fill in the database password, then type in some noise to seed the random number generator.
6. The session key share and corresponding KCV are displayed. Write down both of these.
7. Run the following command to produce an identical transport key; this is generally used within another set of databases which need to use identical transport keys. When this is run, multiple session key shares and KCVs are generated. Write down all of this information.

```
tkstool -I -d . -n verify_transport
```

Responses similar to the following appear:

```
Generating first symmetric key . . .
Generating second symmetric key . . .
Generating third symmetric key . . .
Extracting transport key from operational token . . .
     transport key KCV: A428 53BA
Storing transport key on final specified token . . .
Naming transport key "transport" . . .
Successfully generated, stored, and named the transport key!
```

8. List the contents of the key database again.

```
tkstool -L -d .

 slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
 0 transport
```

9. Use the transport key to generate and wrap a master key, and store the master key in a file called **`file`**.

```
tkstool -W -d . -n wrapped_master -t transport -o file

Enter Password or Pin for "NSS Certificate DB":
Retrieving the transport key (for wrapping) from the specified token . . .
Generating and storing the master key on the specified token . . .
Naming the master key "wrapped_master" . . .
Successfully generated, stored, and named the master key!
Using the transport key to wrap and store the master key . . .
Writing the wrapped data (and resident master key KCV) into the file
called "file" . . .

       wrapped data:   47C0 06DB 7D3F D9ED
                        FE91 7E6F A7E5 91B9
       master key KCV: CED9 4A7B
       (computed KCV of the master key residing inside the wrapped data)
```

10. List the contents of the software key database again.

```
tkstool -L -d .

 slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
 0 wrapped_master
 1 transport
```

NOTE

The order of the keys is not important, and some systems may display the keys in a different order.

11. Use the transport key to generate and unwrap a master key called **`unwrapped_master`** stored in a file called **`file`**.

```
tkstool -U -d . -n unwrapped_master -t transport -i file

Enter Password or Pin for "NSS Certificate DB":
Retrieving the transport key from the specified token (for unwrapping) . . .
Reading in the wrapped data (and resident master key KCV) from the file
called "file" . . .

     wrapped data:   47C0 06DB 7D3F D9ED
                      FE91 7E6F A7E5 91B9
     master key KCV: CED9 4A7B
     (pre-computed KCV of the master key residing inside the wrapped data)

Using the transport key to temporarily unwrap the master key to
recompute its KCV value to check against its pre-computed KCV value . . .
     master key KCV: CED9 4A7B
     (computed KCV of the master key residing inside the wrapped data)
     master key KCV: CED9 4A7B
     (pre-computed KCV of the master key residing inside the wrapped data)

Using the transport key to unwrap and store the master key on the
specified token . . .
Naming the master key "unwrapped_master" . . .
Successfully unwrapped, stored, and named the master key!
```

12. List the contents of the key database to show all keys.

```
tkstool -L -d .

 slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
 0 unwrapped_master
 1 wrapped_master
 2 transport
```

13. Delete a key from the database.

```
tkstool -D -d . -n wrapped_master

Enter Password or Pin for "NSS Certificate DB":
tkstool: 1 key(s) called "wrapped_master" were deleted
```

14. List the contents of the key database again to show all keys.

```
tkstool -L -d .

slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
 0 unwrapped_master
 1 transport
```

Chapter 11. CMCRequest (Creating CMC Requests)

The CMC Request utility, **CMCRequest**, creates a CMC request from one or more PKCS #10 or CRMF requests. The utility can also be used to revoke certificates.

11.1. Syntax

The **CMCRequest** command uses a configuration file (**.cfg**) as a parameter. The **.cfg** file must include the path to the file of the formatted CMC request:

```
CMCRequest /path/to/file.cfg
```

For revocation requests, the **revRequest.enable** parameter must be set to **true**, and related parameters must contain the appropriate information.

The **.cfg** file contains the following parameters:

Parameters	Description
numRequests	The total number of PKCS #10 or CRMF requests. In some cases, the value of this parameter can be 0. For example, **numRequests=1**.
input	The full path and filename of the PKCS #10 or CRMF request, which must be in base-64 encoded format. Multiple filenames are separated by white space. This parameter is a required if the value for **numRequests** is greater than 0. For example, **input=crmf1**.
output	*Required.* The full path and filename for the generated binary CMC request. For example, **output=cmc**.
nickname	*Required.* The nickname of the agent certificate used to sign the full CMC request. For example, **nickname=CS Agent-102504a's 102504a ID**.
dbdir	*Required.* The full path to the directory where the **cert8.db**, **key3.db**, and **secmod.db** databases are located. This is usually the agent's personal directory, such as their browser certificate database in the home directory. For example, **~jsmith/.mozilla/firefox**.
password	*Required.* The token password for **cert8.db**, which stores the agent certificate. For example, **password=secret**.
format	The request format, either **pkcs10** or **crmf**.

For example, **format=crmf**.

The following **.cfg** file parameters set CMC controls:

Parameters	Description
confirmCertAcceptance.enable	If set to **true**, then the request contains this control. If this parameter is not set, the value is assumed to be **false**. For example, **confirmCertAcceptance.enable=false**.
confirmCertAcceptance.serial	The serial number for the **confirmCertAcceptance** control. For example, **confirmCertAcceptance.serial=3**.
confirmCertAcceptance.issuer	The issuer name for the **confirmCertAcceptance** control. For example, **confirmCertAcceptance.issuer=cn=Certificate Manager,ou=102504a,o=102504a,c=us**.
getCert.enable	If set to **true**, then the request contains this attribute. If this parameter is not set, the value is assumed to be **false**. For example, **getCert.enable=false**.
getCert.serial	The serial number for the **getCert** control. For example, **getCert.serial=300**.
getCert.issuer	The issuer name for the **getCert** control. For example, **getCert.issuer=cn=Certificate Manager,ou=102504a,o=102504a,c=us**.
dataReturn.enable	If set to **true**, then the request contains this control. If this parameter is not set, the value is assumed to be **false**. For example, **dataReturn.enable=false**.
dataReturn.data	The data contained in the **dataReturn** control. For example, **dataReturn.data=test**.
transactionMgt.enable	If set to **true**, then the request contains this control. If this parameter is not set, the value is assumed to be **false**. For example, **transactionMgt.enable=true**.

transactionMgt.id	The transaction identifier for **transactionMgt** control. VeriSign recommends that the transaction ID should be an MD5 hash of the public key.
senderNonce.enable	If set to **true**, then the request contains this control. If this parameter is not set, the value is assumed to be **false**. For example, **senderNonce.enable=false**.
senderNonce.id	The ID for the **senderNonce** control. For example, **senderNonce.id=testing**.
revRequest.enable	If set to **true**, then the request contains this control. If this parameter is not set, the value is assumed to be **false**. For example, **revRequest.enable=true**.
revRequest.nickname	The nickname for the certificate being revoked. For example, **revRequest.nickname=newuser's 102504a ID**.
revRequest.issuer	The issuer name for the certificate being revoked. For example, **revRequest.issuer=cn=Certificate Manager,ou=102504a,o=102504a,c=us**.
revRequest.serial	The serial number for the certificate being revoked. For example, **revRequest.serial=75**.
revRequest.reason	The reason for revoking this certificate. The allowed values are **unspecified**, **keyCompromise**, **caCompromise**, **affiliationChanged**, **superseded**, **cessationOfOperation**, **certificateHold**, and **removeFromCRL**. For example, **revRequest.reason=unspecified**.
revRequest.sharedSecret	The shared secret for the revocation request. For example, **revRequest.sharedSecret=testing**.
revRequest.comment	A text comment for the revocation request. For example, **revRequest.comment=readable comment**.
revRequest.invalidityDatePresent	If set to **true**, the current time is the invalidity date for the revoked certificate. If set to **false**, no invalidity date is present.

	For example, **revRequest.invalidityDatePresent=false**.
identityProof.enable	If set to **true**, then the request contains this control. If this parameter is not set, the value is assumed to be **false**. For example, **identityProof.enable=false**.
identityProof.sharedSecret	The shared secret for **identityProof** control. For example, **identityProof.sharedSecret=testing**.
popLinkWitness.enable	If set to **true**, then the request contains this control. If this parameter is not set, the value is assumed to be **false**. For example, **popLinkWitness.enable=false**.
LraPopWitness.enable	If set to **true**, then the request contains this control. If this parameter is not set, the value is assumed to be **false**. For example, **LraPopWitness.enable=false**.
LraPopWitness.bodyPartIDs	The space-delimited list of body part IDs for the **LraPopWtiness** control. For example, **LraPopWitness.bodyPartIDs=1** .

11.2. Usage

Once a simple CMC Request, which contains a PKCS#10 request, has been generated, send it to the CA. The easiest method is to use the end-entities pages:

1. Open the end-entities services pages.

   ```
   https://server.example.com:9444/ca/ee/ca/
   ```

2. Select the **caCMCUserCert** profile.
3. Paste in the CMC request.

Alternatively, use **HttpClient** to post it to the profile.

1. Run the **AtoB** tool to convert the base-64-encoded PKCS #10 request to binary.
2. Use the **HttpClient** utility to send the request.

   There are several profiles where the CMC request can be sent, including **/ca/ee/ca/profileSubmitCMCFull** and **/ca/ee/ca/profileSubmitCMCSimple**. The profile must be specified in the **HttpClient** configuration.

11.3. Output

The **CMCRequest** command generates a certificate request depending on the parameters in a **.cfg** file. The parameters in Example 11.1, “CMC Request .cfg File” are used to create the request in Example 11.2, “CMC Request Output”.

Example 11.1. CMC Request .cfg File

```
#Usage: CMCRequest <configuration file>
#For example, CMCRequest CMCRequest.cfg

#The configuration file should look like as follows:

#numRequests: Total number of PKCS10 requests or CRMF requests.
numRequests=1

#input: full path for the PKCS10 request or CRMF request,
#the content must be in Base-64 encoded format
#Multiple files are supported. They must be separated by space.
#input=pkcs10.i
#input=govReq2.txt
input=myCMC.txt

#output: full path for the CMC request in binary format
output=/tmp/cfu/cmcReq.myCMC

#nickname: nickname for agent certificate which will be used
#to sign the CMC full request.
#nickname=CMS Agent Certificate
#nickname=cfuAgent-ca2's SjcRedhat Domain jaw ca2 ID
nickname=CA Administrator of Instance pki-ca-0124's SjcRedhat Domain 0124 ID

#dbdir: directory for cert8.db, key3.db and secmod.db
dbdir=/tmp/cfu/

#password: password for cert8.db which stores the agent
#certificate
password=netscape

#format: request format, either pkcs10 or crmf
format=crmf

#confirmCertAcceptance.enable: if true, then the request will
#contain this control. Otherwise, false.
confirmCertAcceptance.enable=false

#confirmCertAcceptance.serial: The serial number for
#confirmCertAcceptance control
confirmCertAcceptance.serial=3

#confirmCertAcceptance.issuer: The issuer name for
#confirmCertAcceptance control
confirmCertAcceptance.issuer=cn=Certificate Manager,c=us

#getCert.enable: if true, then the request will contain this
#control. Otherwise, false.
getCert.enable=false

#getCert.serial: The serial number for getCert control
getCert.serial=3

#getCert.issuer: The issuer name for getCert control
getCert.issuer=cn=Certificate Manager,c=us

#dataReturn.enable: if true, then the request will contain
#this control. Otherwise, false.
dataReturn.enable=false
```

```
#dataReturn.data: data contained in the control.
dataReturn.data=test

#transactionMgt.enable: if true, then the request will contain
#this control. Otherwise, false.
transactionMgt.enable=false

#transactionMgt.id: transaction identifier. Verisign recommend
#transactionId to be MD5 hash of publicKey.
transactionMgt.id=

#senderNonce.enable: if true, then the request will contain this
#control. Otherwise, false.
senderNonce.enable=false

#senderNonce.id: sender nonce
senderNonce.id=

#revRequest.enable: if true, then the request will contain this
#control. Otherwise, false.
revRequest.enable=false

#revRequest.nickname: The nickname for the revoke certificate
revRequest.nickname=newuser's 102504a ID

#revRequest.issuer: The issuer name for the certificate being
#revoked.
revRequest.issuer=cn=Certificate Manager,c=us

#revRequest.serial: The serial number for the certificate being
#revoked.
revRequest.serial=61

#revRequest.reason: The reason for revoking this certificate:
#                   unspecified, keyCompromise, caCompromise,
#                   affiliationChanged, superseded, cessationOfOperation,
#                   certificateHold, removeFromCRL
revRequest.reason=unspecified

#revRequest.sharedSecret: The sharedSecret
revRequest.sharedSecret=

#revRequest.comment: The human readable comment
revRequest.comment=

#revRequest.invalidityDatePresent: if true, the current time will be the
#                                  invalidityDate. If false, no invalidityDate
#                                  is present.
revRequest.invalidityDatePresent=false

#identityProof.enable: if true, then the request will contain
#this control. Otherwise, false.
identityProof.enable=false

#identityProof.sharedSecret: Shared Secret
identityProof.sharedSecret=testing

#popLinkWitness.enable:  if true, then the request will contain
#this control. Otherwise, false.
#If you want to test this control, make sure to use CRMFPopClient
```

```
# to generate the CRMF request which will include the
#idPOPLinkWitness attribute in the controls section of the
#CertRequest structure.
popLinkWitness.enable=false

#LraPopWitness.enable: if true, then the request will contain this
#control. Otherwise, false.
LraPopWitness.enable=false

#LraPopWitness.bodyPartIDs: List of body part IDs
#Each id is separated by space.
LraPopWitness.bodyPartIDs=1
```

Example 11.2. CMC Request Output

```
CMCRequest CMCrequest.myCMC.cfg

cert/key prefix =
path = /tmp/cfu/

The CMC enrollment request in base-64 encoded format:

MIIKZwYJKoZIhvcNAQcCoIIKWDCCClQCAQMxCzAJBgUrDgMCGgUAMIIBxAYIKwYB
BQUHDAKgggG2BIIBsjCCAa4wADCCAaShggGgMIIBBgIFAPgzSl8wgceAAQKlDjAM
MQowCAYDVQQDEwF4poGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDhZcSEFI3v
YqNWHsHIH/BDrcVHLuHNuifuSE0fgyirNAwI7IwVReB/I2b1NWSyqh2+9PYIFeSc
VjXvh7p9GU7GmLL4p+Tdpx3YD1JVrumbn6W2uGvMf8UgNx8OxFgkuKy3Z9ohd30x
oTi/hEKoDKxUXN6BY93UPwKLQ7Fpo9RDvQIDAQABqRAwDgYDVR0PAQH/BAQDAgXg
MDMwFQYJKwYBBQUHBQEBDAhyZWdUb2tlbjAaBgkrBgEFBQcFAQIMDWF1dGhlbnRp
Y2F0b3KhgZMwDQYJKoZIhvcNAQEFBQADgYEAtewF4jFndWjpduAzxsxYmBGsPtrE
drCtsm7lvf1ytUPRX0dIEhKgIEQBNsr/UZaCGWrCNpqdKjlSIbsZAw/0Jd8oiRYP
pd6sjYJmBoP5uCf/xft2tJAFDGBAeb3T4VwZb//SasrrRvl6Aa5PBqbh1FrjSCeo
Cc/VeX2nHgwKjj8wADAAoIIHODCCA2owggJSoAMCAQICAQYwDQYJKoZIhvcNAQEL
BQAwUTEeMBwGA1UEChMVU2pjUmVkaGF0IERvbWFpbiAwMTI0MQ8wDQYDVQQLEwZw
a2ktY2ExHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1dGhvcml0eTAeFw0xMTAxMjQy
MzU3MTVaFw0xMzAxMTMyMzU3MTVaMIGJMR4wHAYDVQQKExVTamNSZWRoYXQgRG9t
YWluIDAxMjQxHTAbBgkqhkiG9w0BCQEWDmNmdUByZWRoYXQuY29tMRUwEwYKCZIm
iZPyLGQBARMFYWRtaW4xMTAvBgNVBAMTKENBIEFkbWluaXN0cmF0b3Igb2YgSW5z
dGFuY2UgcGtpLWNhLTAxMjQwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANGu
Qk6xUMkuY8j1/NxXBBEz0N1zZgziqGDMLmQorYxVklDsCMx9tajq3/r9u2CDLaI0
QTvbUwPd1V+CDPfopHG1eTOL62bzLdF1874Q8OW0+UD9m6IFYgnY0toqJJLU/1eO
JUPkbYnGJwmfG3MTWbpr2MrEr+wwalPgmytlaOzxAgMBAAGjgZcwgZQwHwYDVR0j
BBgwFoAU10BlukYi0n1jHqDIvwut/A0qdHswQgYIKwYBBQUHAQEENjA0MDIGCCsG
AQUFBzABhiZodHRwOi8vcGF3LnNqYy5yZWRoYXQuY29tOjkxODAvY2Evb2NzcDAO
BgNVHQ8BAf8EBAMCBPAwHQYDVR0lBBYwFAYIKwYBBQUHAwIGCCsGAQUFBwMEMA0G
CSqGSIb3DQEBCwUAA4IBAQCwQEmjjVmmgEdAO/EYaTQXmfRhEsMYuDium6EoKCpC
Qb4JReUXekxrJnTpTwkUbJq6xiuDozrLHryWAnk1Y6WHxILUkJppCvCiXcVkicvV
eGU2S6p8hKPbC5LLThotN1OIU74N8fdE+zunFV+xnP/4GkJQKuNJiRTZOFmvh/jY
QIqDBcNPhVfcu200H1UaHqLxG22gEByxqs/ma13MEQtaMZBAvicc4i5vhT01YwT2
suYcJDmYpaWVKTjXtm572lNgMYMpNjxnRowicq5Ez8oj5CZc39fB3l3u8fBCRzqo
PlDVQZFzNP+xyvzyJRhUc5oegIaealOdh28X9OXe+eE8MIIDxjCCAq6gAwIBAgIB
ATANBgkqhkiG9w0BAQsFADBRMR4wHAYDVQQKExVTamNSZWRoYXQgRG9tYWluIDAx
MjQxDzANBgNVBAsTBnBraS1jYTEeMBwGA1UEAxMVQ2VydGlmaWNhdGUgQXV0aG9y
aXR5MB4XDTExMDEyNDIzNTYxMloXDTE5MDEyNDIzNTYxMlowUTEeMBwGA1UEChMV
U2pjUmVkaGF0IERvbWFpbiAwMTI0MQ8wDQYDVQQLEwZwa2ktY2ExHjAcBgNVBAMT
FUNlcnRpZmljYXRlIEF1dGhvcml0eTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBANlRZ/b8FFn/8FgVXXg4scSuzTgZ61/upie2zt0n/hY2eMRYh12tlocX
k64WYEREvKAFLF8pYMfoZzldylp9vEWChEWd8OqOM6pcKRpxnphNSOsTlAFh+QbD
rnvusCg63idr4WLiEP92dXZEpIs1m0bCXnKOF2Vio0CX7VM8X2iHQVKOoIQzovsl
Kc+xt/5p/Hy9vFDF+Lyf5dBnT3Rsct/T+Z1pNnHeS5bnv28oxXRdSnnrPPEEVDq2
jj+k1hje4b1aIVuEyGgcKWrlnyZXSei4nY0WDmEv/Lgox6o+QyVEmLMydWj8G5d0
XreQZYke9+XS6OFNah8fFVLW+GCeqtkCAwEAAaOBqDCBpTAfBgNVHSMEGDAWgBTX
QGW6RiLSfWMeoMi/C638DSp0ezAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQE
AwIBxjAdBgNVHQ4EFgQU10BlukYi0n1jHqDIvwut/A0qdHswQgYIKwYBBQUHAQEE
NjA0MDIGCCsGAQUFBzABhiZodHRwOi8vcGF3LnNqYy5yZWRoYXQuY29tOjkxODAv
Y2Evb2NzcDANBgkqhkiG9w0BAQsFAAOCAQEAEfEaydNIzEO6cUEnw9Q3aLf5UcRQ
/K+wggfvtBN33moQD6Z6MmOGiQh/s2bgwDtYgoCnwhkLlpQggZZ2R/Q4b7LV5tzH
B1+v40LZsC4bQ6BPkUIX5gzoCZNJiNlM4Bc+tg92MWIYKj5zHr6yghiJATr87vBY
UxeUOTH7d5i9X6TICsf8AEb50WMFPaoW9GctTwelVYlgg56dFC3wY81bdEBr0SID
l1lW97WuoPU+Jh1OA0AANcYlOh5j9fyOlsqcdUXhPQUsTq2Ou20jpOrh0Aw6CHpQ
3S4rYJSg7MEbI3lQFOapAfOqrl1e3kfgogoIIEQmhOOrjpUnQc+9C7l/gDGCATww
ggE4AgEDMFYwUTEeMBwGA1UEChMVU2pjUmVkaGF0IERvbWFpbiAwMTI0MQ8wDQYD
VQQLEwZwa2ktY2ExHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1dGhvcml0eQIBBjAJ
BgUrDgMCGgUAoD4wFwYJKoZIhvcNAQkDMQoGCCsGAQUFBwwCMCMGCSqGSIb3DQEJ
BDEWBBTJWrAxeErsabiWVokJhrYe8O2AXDANBgkqhkiG9w0BAQEFAASBgJSrhYMo
```

```
smKomXTGaczIjvhYj7IsCUgbpPMqzfhQh5l1X2b5hL3hkWaMDDl9eo2HGZYoE9Lr
6RoIMNs8FCN8F6F8eBzRKlkZTEA+3nXB7gnYVbxrwJrIm2htyTgphu6/yck0wCH9
Og2BekSHQsJ7V7abP04U0VBIUAocJmHwlLnQ


The CMC enrollment request in binary format is stored in /tmp/cfu/cmcReq.myCMC.
```

Chapter 12. CMCEnroll (Performing CMC Enrollments)

The CMC Enrollment utility, **CMCEnroll**, is used to sign a certificate request with an agent's certificate. This can be used in conjunction with the CA end-entity CMC Enrollment form to sign and enroll certificates for users.

12.1. Syntax

This utility has the following syntax:

```
CMCEnroll -d directory_containing_agent_cert -n certificate_nickname -r
certificate_request_file -p certificate_DB_passwd [-c comment]
```

Option	Description
d	The directory containing the **cert8.db**, **key3.db**, and **secmod.db** files associated with the agent certificate. This is usually the agent's personal directory, such as their browser certificate database in the home directory.
n	The nickname of the agent certificate that is used to sign the request.
r	The filename of the certificate request.
p	The password to the NSS certificate database which contains the agent certificate, given in **-d**.

NOTE

Surround values that include spaces with quotation marks.

12.2. Usage

Signed requests must be submitted to the CA to be processed.

1. Create a PKCS #10 certificate request using a tool like **certutil**.
2. Copy the PKCS #10 ASCII output to a text file.
3. Run the **CMCEnroll** command to sign the certificate request. If the input file is **request34.txt**, the agent's certificate is stored in the **~jsmith/.mozilla/firefox** directory, the certificate common name for this CA is **Certificate Manager Agents Cert**, and the password for the certificate database is **1234pass**, the command is as follows:

   ```
   CMCEnroll -d "~jsmith/.mozilla/firefox" -n "Certificate Manager Agents Cert"
   -r "/export/requests/request34.txt" -p "1234pass"
   ```

   The output of this command is stored in a file with the same filename and **.out** appended to the filename.
4. Submit the signed certificate through the CA end-entities page.
   a. Open the end-entities page.
   b. Select the CMC Enrollment profile form.
   c. Paste the content of the output file into the first text area of this form.

d. Remove **-----BEGIN NEW CERTIFICATE REQUEST-----** and **----END NEW CERTIFICATE REQUEST-----** from the pasted content.

e. Select **Certificate Type User Certificate**, fill in the contact information, and submit the form.

5. The certificate is immediately processed and returned since a signed request was sent and the **CMCAuth** plug-in was enabled.
6. Use the agent page to search for the new certificate.

12.3. Output

CMCEnroll takes a standard PKCS #10 certificate request and signs it with an agent certificate. The output is also a certificate request which can be submitted through the appropriate profile.

```
# CMCEnroll -d ~jsmith/.mozilla/firefox -n "CA Administrator of Instance pki-ca
Example Domain ID" -r pkcs10.i -p secret
cert/key prefix =
path = .
-----BEGIN NEW CERTIFICATE REQUEST-----
MIILsQYJKoZIhvcNAQcCoIILojCCC54CAQMxCzAJBgUrDgMCGgUAMIIDDgYIKwYB
BQUHDAKgggMABIIC/DCCAvgwUzAtAgECBggrBgEFBQcHBjEeBBxlWXlObThoN1dk
eDNHcEE1MllxcERTNXVRVlE9MCICAQMGCCsGAQUFBwcFMRMCEQD6Yglp1fV/P0VA
CzSQPuO9MIICm6CCApcCAQEwggKQMIIBeAIBADBLMSUwIwYDVQQKExxTZmJheSBS
ZWRoYXQgRG9tYWluIHBhdyAxMjEyMSIwIAYDVQQDExlEUk0gU3Vic3lzdGVtIENl
cnRpZmljYXRlMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqZvFwMGg
f2W0/OAlPBLcCq5dlHkrrQIsKx4P2c1Q+8eFJAxOPLaA5xVs+AGHMUGeBTsRuk0Q
26NRWq9JOtrRbIaUFy4ilTI7vpM2NThObMidE8EUs8p4wef5C0kTgkKA1/B+F6IF
JU8LVMtF2hHOTUEPheRMyb4fTJPjUec+zNduKe2xdlrJmmUgCC8ibC4KavyKDe99
T89CcXNZe/NgyejILhZ95wPzaW7xmN36fXUrxsMDKLJSn4X2i5DlKyuybw4+rosv
jI2J9o2Kfj1EDHHkzC57iYjNF4wmgUyZobnSYNyNLZWzbXiGEHq8/yWaxZxML4fP
SYspU2EIWh0L/QIDAQABoAAwDQYJKoZIhvcNAQEEBQADggEBAEJWi4aUqFJ4uEi3
xwf9bAdPhRpifdfWZN1BfVQn5DoaI3Bhm4EDsVuugm7E4IIls1SysmsQhk2F5A3p
ltpGCnoMXnFg9ToCpCu6wKi9xRwT5+2Iwbjnj1JtMkQx90xebBQ9v3+VVClRXjJs
edp3OSDIB/LA5/ieg+be/VbWaTb1yrvGaZruZT+iu2GPGQ8R40uPM3n22K/Vu3LI
dhO5uNIP8zAaEpda16detBXievCZgZsuC8iRkKfrRir97L2zJk46/xMgno6CqQFL
4p9gH5/G/vddH2pbNI8kqoIEthmSw6mD/yQjaSAsAC8DM8wWAo4SIz6MXTcMPGgO
moDuVVowADAAoIIHODCCA2owggJSoAMCAQICAQYwDQYJKoZIhvcNAQELBQAwUTEe
MBwGA1UEChMVU2pjUmVkaGF0IERvbWFpbiAwMTI0MQ8wDQYDVQQLEwZwa2ktY2Ex
HjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1dGhvcml0eTAeFw0xMTAxMjQyMzU3MTVa
Fw0xMzAxMTMyMzU3MTVaMIGJMR4wHAYDVQQKExVTamNSZWRoYXQgRG9tYWluIDAx
MjQxHTAbBgkqhkiG9w0BCQEWDmNmdUByZWRoYXQuY29tMRUwEwYKCZImiZPyLGQB
ARMFYWRtaW4xMTAvBgNVBAMTKENBIEFkbWluaXN0cmF0b3Igb2YgSW5zdGFuY2Ug
cGtpLWNhLTAxMjQwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANGuQk6xUMku
Y8j1/NxXBBEz0N1zZgziqGDMLmQorYxVklDsCMx9tajq3/r9u2CDLaI0QTvbUwPd
1V+CDPfopHG1eTOL62bzLdF1874Q8OW0+UD9m6IFYgnY0toqJJLU/1eOJUPkbYnG
JwmfG3MTWbpr2MrEr+wwalPgmytlaOzxAgMBAAGjgZcwgZQwHwYDVR0jBBgwFoAU
10BlukYi0n1jHqDIvwut/A0qdHswQgYIKwYBBQUHAQEENjA0MDIGCCsGAQUFBzAB
hiZodHRwOi8vcGF3LnNqYy5yZWRoYXQuY29tOjkxODAvY2Evb2NzcDAOBgNVHQ8B
Af8EBAMCBPAwHQYDVR0lBBYwFAYIKwYBBQUHAwIGCCsGAQUFBwMEMA0GCSqGSIb3
DQEBCwUAA4IBAQCwQEmjjVmmgEdAO/EYaTQXmfRhEsMYuDium6EoKCpCQb4JReUX
ekxrJnTpTwkUbJq6xiuDozrLHryWAnk1Y6WHxILUkJppCvCiXcVkicvVeGU2S6p8
hKPbC5LLThotN1OIU74N8fdE+zunFV+xnP/4GkJQKuNJiRTZOFmvh/jYQIqDBcNP
hVfcu200H1UaHqLxG22gEByxqs/ma13MEQtaMZBAvicc4i5vhT01YwT2suYcJDmY
paWVKTjXtm572lNgMYMpNjxnRowicq5Ez8oj5CZc39fB3l3u8fBCRzqoPlDVQZFz
NP+xyvzyJRhUc5oegIaealOdh28X9OXe+eE8MIIDxjCCAq6gAwIBAgIBATANBgkq
hkiG9w0BAQsFADBRMR4wHAYDVQQKExVTamNSZWRoYXQgRG9tYWluIDAxMjQxDzAN
BgNVBAsTBnBraS1jYTEeMBwGA1UEAxMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4X
DTExMDEyNDIzNTYxMloXDTE5MDEyNDIzNTYxMlowUTEeMBwGA1UEChMVU2pjUmVk
aGF0IERvbWFpbiAwMTI0MQ8wDQYDVQQLEwZwa2ktY2ExHjAcBgNVBAMTFUNlcnRp
ZmljYXRlIEF1dGhvcml0eTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
ANlRZ/b8FFn/8FgVXXg4scSuzTgZ61/upie2zt0n/hY2eMRYh12tlocXk64WYERE
vKAFLF8pYMfoZzldylp9vEWChEWd8OqOM6pcKRpxnphNSOsTlAFh+QbDrnvusCg6
3idr4WLiEP92dXZEpIs1m0bCXnKOF2Vio0CX7VM8X2iHQVKOoIQzovslKc+xt/5p
/Hy9vFDF+Lyf5dBnT3Rsct/T+Z1pNnHeS5bnv28oxXRdSnnrPPEEVDq2jj+k1hje
4b1aIVuEyGgcKWrlnyZXSei4nY0WDmEv/Lgox6o+QyVEmLMydWj8G5d0XreQZYke
9+XS6OFNah8fFVLW+GCeqtkCAwEAAaOBqDCBpTAfBgNVHSMEGDAWgBTXQGW6RiLS
fWMeoMi/C638DSp0ezAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBxjAd
BgNVHQ4EFgQU10BlukYi0n1jHqDIvwut/A0qdHswQgYIKwYBBQUHAQEENjA0MDIG
CCsGAQUFBzABhiZodHRwOi8vcGF3LnNqYy5yZWRoYXQuY29tOjkxODAvY2Evb2Nz
cDANBgkqhkiG9w0BAQsFAAOCAQEAEfEaydNIzEO6cUEnw9Q3aLf5UcRQ/K+wggfv
tBN33moQD6Z6MmOGiQh/s2bgwDtYgoCnwhkLlpQggZZ2R/Q4b7LV5tzHB1+v40LZ
sC4bQ6BPkUIX5gzoCZNJiNlM4Bc+tg92MWIYKj5zHr6yghiJATr87vBYUxeUOTH7
d5i9X6TICsf8AEb50WMFPaoW9GctTwelVYlgg56dFC3wY81bdEBr0SIDl1lW97Wu
oPU+Jh1OA0AANcYlOh5j9fyOlsqcdUXhPQUsTq2Ou20jpOrh0Aw6CHpQ3S4rYJSg
```

```
7MEbI3lQFOapAfOqrl1e3kfgogoIIEQmhOOrjpUnQc+9C7l/gDGCATwwggE4AgED
MFYwUTEeMBwGA1UEChMVU2pjUmVkaGF0IERvbWFpbiAwMTI0MQ8wDQYDVQQLEwZw
a2ktY2ExHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1dGhvcml0eQIBBjAJBgUrDgMC
GgUAoD4wFwYJKoZIhvcNAQkDMQoGCCsGAQUFBwwCMCMGCSqGSIb3DQEJBDEWBBQa
be8HVhF1o9FP9E2tjfOMkdmuQDANBgkqhkiG9w0BAQEFAASBgDAXIZgfFWv9GcLd
+yRET7HOgi1uTAiWwRoV2KZ0hbDowuNsaMHu5k4oic8wcOCOyAvWS/9o64RPG96f
QKLHrAa8CJ3hqzDl3xaOuNF/iJWM4R47136DRhW9cCA1qFH3FyiUcwBdGd5R8DrE
r+ce2ZSTtI2Jpif83w7Ro5VqSMMN
-----END NEW CERTIFICATE REQUEST-----
```

Chapter 13. CMCResponse (Parsing a CMC Response)

The CMC Response utility, **CMCResponse**, parses a CMC response received by the utility.

13.1. Syntax

The CMC Response utility uses the following syntax:

```
CMCResponse -d directoryName -i /path/to/CMCResponse.file
```

Options	Description
d	Specifies the path to the **cert8.db** directory. This is usually the agent's personal directory, such as their browser certificate database in the home directory.
i	Specifies the path and filename of the CMC response file.

The parsed output is printed to the screen.

13.2. Usage and Output

The entire purpose of **CMCResponse** is to parse a CMC response. As explained in Chapter 11, *CMCRequest (Creating CMC Requests)*, a CMC request is generated and then submitted to a CMC profile which returns a response in the CMC format. In one common use case, a tool like **HttpClient** is used to submit a request and then retrieve the response. That response is sent to **CMCResponse** to parse.

The first step is to create the **.cfg** file which will be used to by **HttpClient** to submit the request.

```
#host: host name for the http server
host=server.example.com

#port: port number
port=9444

#secure: true for secure connection, false for nonsecure connection
secure=true

#input: full path for the enrollment request, the content must be in binary format
input=/tmp/cfu/cmcReq.myCMC

#output: full path for the response in binary format
output=/tmp/cfu/cmcResponse.myCMC

#dbdir: directory for cert8.db, key3.db and secmod.db
#This parameter will be ignored if secure=false
dbdir=/tmp/cfu

#clientmode: true for client authentication, false for no client authentication
#This parameter will be ignored if secure=false
clientmode=false

#password: password for cert8.db
#This parameter will be ignored if secure=false and clientauth=false
password=netscape

#nickname: nickname for client certificate
#This parameter will be ignored if clientmode=false
nickname=

#servlet: servlet name
servlet=/ca/ee/ca/profileSubmitCMCFull
```

That configuration file is then passed to **`HttpClient`**, which received the binary CMC response.

```
# HttpClient HttpClient.cfg

Total number of bytes read = 2667
handshake happened
Total number of bytes read = 2287
MIII6wYJKoZIhvcNAQcCoIII3DCCCNgCAQMxDjAMBghghkgBZQMEAQUAMDUGCCsG
AQUFBwwDoCkEJzAlMB8wHQIBAQYIKwYBBQUHBwExDjAMAgEAMAcCBQD4M0pfMAAw
AKCCBrowggLsMIIB1KADAgECAgEaMA0GCSqGSIb3DQEBCwUAMFExHjAcBgNVBAoT
FVNqY1JlZGhhdCBEb21haW4gMDEyNDEPMA0GA1UECxMGcGtpLWNhMR4wHAYDVQQD
ExVDZXJ0aWZpY2F0ZSBBdXRob3JpdHkwHhcNMTEwMzA4MTY0MTMwWhcNMTEwOTA0
MTY0MTMwWjAMMQowCAYDVQQDEwF4MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDhZcSEFI3vYqNWHsHIH/BDrcVHLuHNuifuSE0fgyirNAwI7IwVReB/I2b1NWSy
qh2+9PYIFeScVjXvh7p9GU7GmLL4p+Tdpx3YD1JVrumbn6W2uGvMf8UgNx8OxFgk
uKy3Z9ohd30xoTi/hEKoDKxUXN6BY93UPwKLQ7Fpo9RDvQIDAQABo4GXMIGUMB8G
A1UdIwQYMBaAFNdAZbpGItJ9Yx6gyL8LrfwNKnR7MEIGCCsGAQUFBwEBBDYwNDAy
BggrBgEFBQcwAYYmaHR0cDovL3Bhdy5zamMucmVkaGF0LmNvbTo5MTgwL2NhL29j
c3AwDgYDVR0PAQH/BAQDAgXgMB0GA1UdJQQWMBQGCCsGAQUFBwMCBggrBgEFBQcD
BDANBgkqhkiG9w0BAQsFAAOCAQEAQxdBWvoc5/0SKUGdWvhs4NPqU1cX4fjjUW8t
famLXyk37K7PZM/f4wIso37OuQUQO/tuGR0+8EoBD8NfFJwGcMLb1XIfR/2n/Ndq
TmT6qRnuCST4ucQBEtE8rYkFYZQ5Z22N8QPBjiNvoO5qs8X9xMzmbJrjSyNwGJHl
UBDLhyqgVLzdl80UycoFQPPp8vi4/+2/e1+FFRUjtGgNE1Yc5DdrTeST3h5nA/uS
htQRHj8fzSjE/07zEyMFc/IAmCV3xWkiQK2uHJBrYBKFYVEZ7YJQ6sO/q/lUdv3H
5x6YqEWMqqEJhxru6PRhHKU8WeECu+Z5O+wfIa7BOCjz+AVvLDCCA8YwggKuoAMC
AQICAQEwDQYJKoZIhvcNAQELBQAwUTEeMBwGA1UEChMVU2pjUmVkaGF0IERvbWFp
biAwMTI0MQ8wDQYDVQQLEwZwa2ktY2ExHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1
dGhvcml0eTAeFw0xMTAxMjQyMzU2MTJaFw0xOTAxMjQyMzU2MTJaMFExHjAcBgNV
BAoTFVNqY1JlZGhhdCBEb21haW4gMDEyNDEPMA0GA1UECxMGcGtpLWNhMR4wHAYD
VQQDExVDZXJ0aWZpY2F0ZSBBdXRob3JpdHkwggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQDZUWf2/BRZ//BYFV14OLHErs04Getf7qYnts7dJ/4WNnjEWIdd
rZaHF5OuFmBERLygBSxfKWDH6Gc5XcpafbxFgoRFnfDqjjOqXCkacZ6YTUjrE5QB
YfkGw6577rAoOt4na+Fi4hD/dnV2RKSLNZtGwl5yjhdlYqNAl+1TPF9oh0FSjqCE
M6L7JSnPsbf+afx8vbxQxfi8n+XQZ090bHLf0/mdaTZx3kuW579vKMV0XUp56zzx
BFQ6to4/pNYY3uG9WiFbhMhoHClq5Z8mV0nouJ2NFg5hL/y4KMeqPkMlRJizMnVo
/BuXdF63kGWJHvfl0ujhTWofHxVS1vhgnqrZAgMBAAGjgagwgaUwHwYDVR0jBBgw
FoAU10BlukYi0n1jHqDIvwut/A0qdHswDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8B
Af8EBAMCAcYwHQYDVR0OBBYEFNdAZbpGItJ9Yx6gyL8LrfwNKnR7MEIGCCsGAQUF
BwEBBDYwNDAyBggrBgEFBQcwAYYmaHR0cDovL3Bhdy5zamMucmVkaGF0LmNvbTo5
MTgwL2NhL29jc3AwDQYJKoZIhvcNAQELBQADggEBABHxGsnTSMxDunFBJ8PUN2i3
+VHEUPyvsIIH77QTd95qEA+mejJjhokIf7Nm4MA7WIKAp8IZC5aUIIGWdkf0OG+y
1ebcxwdfr+NC2bAuG0OgT5FCF+YM6AmTSYjZTOAXPrYPdjFiGCo+cx6+soIYiQE6
/O7wWFMXlDkx+3eYvV+kyArH/ABG+dFjBT2qFvRnLU8HpVWJYIOenRQt8GPNW3RA
a9EiA5dZVve1rqD1PiYdTgNAADXGJToeY/X8jpbKnHVF4T0FLE6tjrttI6Tq4dAM
Ogh6UN0uK2CUoOzBGyN5UBTmqQHzqq5dXt5H4KIKCCBEJoTjq46VJ0HPvQu5f4Ax
ggHMMIIByAIBAzBWMFExHjAcBgNVBAoTFVNqY1JlZGhhdCBEb21haW4gMDEyNDEP
MA0GA1UECxMGcGtpLWNhMR4wHAYDVQQDExVDZXJ0aWZpY2F0ZSBBdXRob3JpdHkC
AQEwDAYIYIZIAWUDBAEFAKBKMBcGCSqGSIb3DQEJAzEKBggrBgEFBQcMAzAvBgkq
hkiG9w0BCQQxIgQgXUsQ5rl+G2aiKpAp68LLdF7uOcPDOYbWlacKxpwKfZIwDQYJ
KoZIhvcNAQEBBQAEggEAa4fQfye0ogzxpFYZd98JNZlTuWeluDBv+HwZeIaRWYn4
/YlbZyn98gBaX5V1NNXsmRO1D8iKa7O+4XORweFnEdzqLDQCzN/TFsnKqT8dYHQT
iY4kd2msBOqYa+x3ZKZoEGvRlPMCRXBMTKfSmq963NT7hCZyLA2jmATs4eYrNyQp
xHPzxrUy0Ftj/NJKNb6g3JtSinUp9RkNMArAyg0ORFCcRbCRQNmxYIFkTyE7/yVY
uaRyE7XIPoBqdo5BWgsQlD7GxK0PeSzTBoqmygLu7gZZfx7pghV4YrXIiYtgMafA
GQwiK2Jj1zs/eRR3MN3TvhSYTzavNxq7MXGQVavLQQ==

The response in binary format is stored in /tmp/jsmith/cmcResponse.myCMC
```

The last part of the **`HttpClient`** response shows where the CMC response file is located, and that file can be used by **`CMCResponse`**. When **`CMCResponse`** parses the file, it shows the pretty-print version of the response.

```
# CMCResponse -d . -i cmcResponse.myCMC
Certificates:
    Certificate:
        Data:
            Version:  v3
            Serial Number: 0x1A
            Signature Algorithm: SHA256withRSA - 1.2.840.113549.1.1.11
            Issuer: CN=Certificate Authority,OU=pki-ca,O=SjcRedhat Domain 0124
            Validity:
                Not Before: Tuesday, March 8, 2011 8:41:30 AM PST
America/Los_Angeles
                Not  After: Sunday, September 4, 2011 9:41:30 AM PDT
America/Los_Angeles
            Subject: CN=x
            Subject Public Key Info:
                Algorithm: RSA - 1.2.840.113549.1.1.1
                Public Key:
                    Exponent: 65537
                    Public Key Modulus: (1024 bits) :
                        E1:65:C4:84:14:8D:EF:62:A3:56:1E:C1:C8:1F:F0:43:
                        AD:C5:47:2E:E1:CD:BA:27:EE:48:4D:1F:83:28:AB:34:
                        0C:08:EC:8C:15:45:E0:7F:23:66:F5:35:64:B2:AA:1D:
                        BE:F4:F6:08:15:E4:9C:56:35:EF:87:BA:7D:19:4E:C6:
                        98:B2:F8:A7:E4:DD:A7:1D:D8:0F:52:55:AE:E9:9B:9F:
                        A5:B6:B8:6B:CC:7F:C5:20:37:1F:0E:C4:58:24:B8:AC:
                        B7:67:DA:21:77:7D:31:A1:38:BF:84:42:A8:0C:AC:54:
                        5C:DE:81:63:DD:D4:3F:02:8B:43:B1:69:A3:D4:43:BD
            Extensions:
                Identifier: Authority Key Identifier - 2.5.29.35
                    Critical: no
                    Key Identifier:
                        D7:40:65:BA:46:22:D2:7D:63:1E:A0:C8:BF:0B:AD:FC:
                        0D:2A:74:7B
                Identifier: 1.3.6.1.5.5.7.1.1
                    Critical: no
                    Value:
                        30:34:30:32:06:08:2B:06:01:05:05:07:30:01:86:26:
                        68:74:74:70:3A:2F:2F:70:61:77:2E:73:6A:63:2E:72:
                        65:64:68:61:74:2E:63:6F:6D:3A:39:31:38:30:2F:63:
                        61:2F:6F:63:73:70
                Identifier: Key Usage: - 2.5.29.15
                    Critical: yes
                    Key Usage:
                        Digital Signature
                        Non Repudiation
                        Key Encipherment
                Identifier: Extended Key Usage: - 2.5.29.37
                    Critical: no
                    Extended Key Usage:
                        1.3.6.1.5.5.7.3.2
                        1.3.6.1.5.5.7.3.4
        Signature:
            Algorithm: SHA256withRSA - 1.2.840.113549.1.1.11
            Signature:
                43:17:41:5A:FA:1C:E7:FD:12:29:41:9D:5A:F8:6C:E0:
                D3:EA:53:57:17:E1:F8:E3:51:6F:2D:7D:A9:8B:5F:29:
                37:EC:AE:CF:64:CF:DF:E3:02:2C:A3:7E:CE:B9:05:10:
                3B:FB:6E:19:1D:3E:F0:4A:01:0F:C3:5F:14:9C:06:70:
                C2:DB:D5:72:1F:47:FD:A7:FC:D7:6A:4E:64:FA:A9:19:
                EE:09:24:F8:B9:C4:01:12:D1:3C:AD:89:05:61:94:39:
```

```
                67:6D:8D:F1:03:C1:8E:23:6F:A0:EE:6A:B3:C5:FD:C4:
                CC:E6:6C:9A:E3:4B:23:70:18:91:E5:50:10:CB:87:2A:
                A0:54:BC:DD:97:CD:14:C9:CA:05:40:F3:E9:F2:F8:B8:
                FF:ED:BF:7B:5F:85:15:15:23:B4:68:0D:13:56:1C:E4:
                37:6B:4D:E4:93:DE:1E:67:03:FB:92:86:D4:11:1E:3F:
                1F:CD:28:C4:FF:4E:F3:13:23:05:73:F2:00:98:25:77:
                C5:69:22:40:AD:AE:1C:90:6B:60:12:85:61:51:19:ED:
                82:50:EA:C3:BF:AB:F9:54:76:FD:C7:E7:1E:98:A8:45:
                8C:AA:A1:09:87:1A:EE:E8:F4:61:1C:A5:3C:59:E1:02:
                BB:E6:79:3B:EC:1F:21:AE:C1:38:28:F3:F8:05:6F:2C
        FingerPrint
    Certificate:
        Data:
            Version:  v3
            Serial Number: 0x1
            Signature Algorithm: SHA256withRSA - 1.2.840.113549.1.1.11
            Issuer: CN=Certificate Authority,OU=pki-ca,O=SjcRedhat Domain 0124
            Validity:
                Not Before: Monday, January 24, 2011 3:56:12 PM PST
America/Los_Angeles
                Not  After: Thursday, January 24, 2019 3:56:12 PM PST
America/Los_Angeles
            Subject: CN=Certificate Authority,OU=pki-ca,O=SjcRedhat Domain 0124
            Subject Public Key Info:
                Algorithm: RSA - 1.2.840.113549.1.1.1
                Public Key:
                    Exponent: 65537
                    Public Key Modulus: (2048 bits) :
                        D9:51:67:F6:FC:14:59:FF:F0:58:15:5D:78:38:B1:C4:
                        AE:CD:38:19:EB:5F:EE:A6:27:B6:CE:DD:27:FE:16:36:
                        78:C4:58:87:5D:AD:96:87:17:93:AE:16:60:44:44:BC:
                        A0:05:2C:5F:29:60:C7:E8:67:39:5D:CA:5A:7D:BC:45:
                        82:84:45:9D:F0:EA:8E:33:AA:5C:29:1A:71:9E:98:4D:
                        48:EB:13:94:01:61:F9:06:C3:AE:7B:EE:B0:28:3A:DE:
                        27:6B:E1:62:E2:10:FF:76:75:76:44:A4:8B:35:9B:46:
                        C2:5E:72:8E:17:65:62:A3:40:97:ED:53:3C:5F:68:87:
                        41:52:8E:A0:84:33:A2:FB:25:29:CF:B1:B7:FE:69:FC:
                        7C:BD:BC:50:C5:F8:BC:9F:E5:D0:67:4F:74:6C:72:DF:
                        D3:F9:9D:69:36:71:DE:4B:96:E7:BF:6F:28:C5:74:5D:
                        4A:79:EB:3C:F1:04:54:3A:B6:8E:3F:A4:D6:18:DE:E1:
                        BD:5A:21:5B:84:C8:68:1C:29:6A:E5:9F:26:57:49:E8:
                        B8:9D:8D:16:0E:61:2F:FC:B8:28:C7:AA:3E:43:25:44:
                        98:B3:32:75:68:FC:1B:97:74:5E:B7:90:65:89:1E:F7:
                        E5:D2:E8:E1:4D:6A:1F:1F:15:52:D6:F8:60:9E:AA:D9
            Extensions:
                Identifier: Authority Key Identifier - 2.5.29.35
                    Critical: no
                    Key Identifier:
                        D7:40:65:BA:46:22:D2:7D:63:1E:A0:C8:BF:0B:AD:FC:
                        0D:2A:74:7B
                Identifier: Basic Constraints - 2.5.29.19
                    Critical: yes
                    Is CA: yes
                    Path Length Constraint: UNLIMITED
                Identifier: Key Usage: - 2.5.29.15
                    Critical: yes
                    Key Usage:
                        Digital Signature
                        Non Repudiation
                        Key CertSign
```

```
                        Crl Sign
                Identifier: Subject Key Identifier - 2.5.29.14
                    Critical: no
                    Key Identifier:
                        D7:40:65:BA:46:22:D2:7D:63:1E:A0:C8:BF:0B:AD:FC:
                        0D:2A:74:7B
                Identifier: 1.3.6.1.5.5.7.1.1
                    Critical: no
                    Value:
                        30:34:30:32:06:08:2B:06:01:05:05:07:30:01:86:26:
                        68:74:74:70:3A:2F:2F:70:61:77:2E:73:6A:63:2E:72:
                        65:64:68:61:74:2E:63:6F:6D:3A:39:31:38:30:2F:63:
                        61:2F:6F:63:73:70
        Signature:
            Algorithm: SHA256withRSA - 1.2.840.113549.1.1.11
            Signature:
                11:F1:1A:C9:D3:48:CC:43:BA:71:41:27:C3:D4:37:68:
                B7:F9:51:C4:50:FC:AF:B0:82:07:EF:B4:13:77:DE:6A:
                10:0F:A6:7A:32:63:86:89:08:7F:B3:66:E0:C0:3B:58:
                82:80:A7:C2:19:0B:96:94:20:81:96:76:47:F4:38:6F:
                B2:D5:E6:DC:C7:07:5F:AF:E3:42:D9:B0:2E:1B:43:A0:
                4F:91:42:17:E6:0C:E8:09:93:49:88:D9:4C:E0:17:3E:
                B6:0F:76:31:62:18:2A:3E:73:1E:BE:B2:82:18:89:01:
                3A:FC:EE:F0:58:53:17:94:39:31:FB:77:98:BD:5F:A4:
                C8:0A:C7:FC:00:46:F9:D1:63:05:3D:AA:16:F4:67:2D:
                4F:07:A5:55:89:60:83:9E:9D:14:2D:F0:63:CD:5B:74:
                40:6B:D1:22:03:97:59:56:F7:B5:AE:A0:F5:3E:26:1D:
                4E:03:40:00:35:C6:25:3A:1E:63:F5:FC:8E:96:CA:9C:
                75:45:E1:3D:05:2C:4E:AD:8E:BB:6D:23:A4:EA:E1:D0:
                0C:3A:08:7A:50:DD:2E:2B:60:94:A0:EC:C1:1B:23:79:
                50:14:E6:A9:01:F3:AA:AE:5D:5E:DE:47:E0:A2:0A:08:
                20:44:26:84:E3:AB:8E:95:27:41:CF:BD:0B:B9:7F:80
        FingerPrint


Number of controls is 1
Control #0: CMCStatusInfo
   OID: {1 3 6 1 5 5 7 7 1}
   BodyList: 4164110943
   Status: SUCCESS
```

Chapter 14. CMCRevoke (Signing a Revocation Request)

The CMC Revocation utility, **CMCRevoke**, signs a revocation request with an agent's certificate.

14.1. Syntax

This utility has the following syntax:

```
CMCRevoke -ddirectoryName -hpassword -nnickname -iissuerName -sserialNumber  -
mreasonToRevoke -ccomment
```

IMPORTANT

Do not have a space between the argument and its value. For example, giving a serial number of 26 is **-s26**, not **-s 26**.

NOTE

Surround values that include spaces in quotation marks.

Option	Description
c	Text comments about the request.
d	The path to the directory where the **cert8.db**, **key3.db**, and **secmod.db** databases containing the agent certificates are located. This is usually the agent's personal directory, such as their browser certificate database in the home directory.
h	The password to access the NSS database containing the agent's certificate.
i	The issuer name of the certificate being revoked.
m	The reason the certificate is being revoked. The reason code for the different allowed revocation reasons are as follows: - *0* - Unspecified. - *1* - Key compromised. - *2* - CA key compromised. - *3* - Affiliation changed. - *4* - Certificate superseded. - *5* - Cessation of operation. - *6* - Certificate is on hold.
n	The nickname of the agent's certificate.
s	The decimal serial number of the certificate being revoked.

NOTE

Surround values that include spaces in quotation marks.

14.2. Testing CMC Revocation

Test that CMC revocation is working properly by doing the following:

1. Create a CMC revocation request for an existing certificate. For example, if the directory containing the agent certificate is **~jsmith/.mozilla/firefox/**, the nickname of the certificate is **CertificateManagerAgentCert**, and the serial number of the certificate is **22**, the command is as follows:

```
CMCRevoke -d"~jsmith/.mozilla/firefox/" -n"Certificate Manager Agent Cert" -
i"cn=agentAuthMgr" -s22 -m0 -c"test comment"
```

2. Open the CA's end-entities page.
3. Select the **Revocation** tab.
4. Select the **CMC Revoke** link in the menu.
5. Paste the output from the **CMCRevoke** operation into the text box. Remove the **-----BEGIN NEW CERTIFICATE REQUEST-----** and **----END NEW CERTIFICATE REQUEST-----** lines from the pasted content.
6. Click **Submit**.
7. The results page displays that certificate 22 has been revoked.

14.3. Output

CMCRevoke generates a revocation request that is signed by an agent's certificate, so the output returned is in the format of a certificate *request*. For example:

```
# CMCRevoke -d"~jsmith/.mozilla/firefox" -n"CA Administrator of Instance pki-ca
Example Domain ID" -i"CN=Certificate Authority,OU=pki-ca,O=Example Domain" -s22 -
m6 -hsecret -c"test comment"
cert/key prefix =
path = .
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIJcgYJKoZIhvcNAQcCoIIJYzCCCV8CAQMxCzAJBgUrDgMCGgUAMIHQBggrBgEF
BQcMAqCBwwSBwDCBvTCBtDAtAgEBBggrBgEFBQcHBjEeBBxNZ3l1eFFzc0VWRDJW
SlE4U0VuVFVXYzEvSjg9MIGCAgECBggrBgEFBQcHETFzMHEwUTEeMBwGA1UEChMV
U2pjUmVkaGF0IERvbWFpbiAwMTI0MQ8wDQYDVQQLEwZwa2ktY2ExHjAcBgNVBAMT
FUNlcnRpZmljYXRlIEF1dGhvcml0eQIBFgoBBgQIbmV0c2NhcGUMDHRlc3QgY29t
bWVudDAAMAAwAKCCBzgwggNqMIICUqADAgECAgEGMA0GCSqGSIb3DQEBCwUAMFEx
HjAcBgNVBAoTFVNqY1JlZGhhdCBEb21haW4gMDEyNDEPMA0GA1UECxMGcGtpLWNh
MR4wHAYDVQQDExVDZXJ0aWZpY2F0ZSBBdXRob3JpdHkwHhcNMTEwMTI0MjM1NzE1
WhcNMTMwMTEzMjM1NzE1WjCBiTEeMBwGA1UEChMVU2pjUmVkaGF0IERvbWFpbiAw
MTI0MR0wGwYJKoZIhvcNAQkBFg5jZnVAcmVkaGF0LmNvbTEVMBMGCgmSJomT8ixk
AQETBWFkbWluMTEwLwYDVQQDEyhDQSBBZG1pbmlzdHJhdG9yIG9mIEluc3RhbmNl
IHBraS1jYS0wMTI0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDRrkJOsVDJ
LmPI9fzcVwQRM9Ddc2YM4qhgzC5kKK2MVZJQ7AjMfbWo6t/6/btggy2iNEE721MD
3dVfggz36KRxtXkzi+tm8y3RdfO+EPDltPlA/ZuiBWIJ2NLaKiSS1P9XjiVD5G2J
xicJnxtzE1m6a9jKxK/sMGpT4JsrZWjs8QIDAQABo4GXMIGUMB8GA1UdIwQYMBaA
FNdAZbpGItJ9Yx6gyL8LrfwNKnR7MEIGCCsGAQUFBwEBBDYwNDAyBggrBgEFBQcw
AYYmaHR0cDovL3Bhdy5zamMucmVkaGF0LmNvbTo5MTgwL2NhL29jc3AwDgYDVR0P
AQH/BAQDAgTwMB0GA1UdJQQWMBQGCCsGAQUFBwMCBggrBgEFBQcDBDANBgkqhkiG
9w0BAQsFAAOCAQEAsEBJo41ZpoBHQDvxGGk0F5n0YRLDGLg4rpuhKCgqQkG+CUXl
F3pMayZ06U8JFGyausYrg6M6yx68lgJ5NWOlh8SC1JCaaQrwol3FZInL1XhlNkuq
fISj2wuSy04aLTdTiFO+DfH3RPs7pxVfsZz/+BpCUCrjSYkU2ThZr4f42ECKgwXD
T4VX3LttNB9VGh6i8RttoBAcsarP5mtdzBELWjGQQL4nHOIub4U9NWME9rLmHCQ5
mKWllSk417Zue9pTYDGDKTY8Z0aMInKuRM/KI+QmXN/Xwd5d7vHwQkc6qD5Q1UGR
czT/scr88iUYVHOaHoCGnmpTnYdvF/Tl3vnhPDCCA8YwggKuoAMCAQICAQEwDQYJ
KoZIhvcNAQELBQAwUTEeMBwGA1UEChMVU2pjUmVkaGF0IERvbWFpbiAwMTI0MQ8w
DQYDVQQLEwZwa2ktY2ExHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1dGhvcml0eTAe
Fw0xMTAxMjQyMzU2MTJaFw0xOTAxMjQyMzU2MTJaMFExHjAcBgNVBAoTFVNqY1Jl
ZGhhdCBEb21haW4gMDEyNDEPMA0GA1UECxMGcGtpLWNhMR4wHAYDVQQDExVDZXJ0
aWZpY2F0ZSBBdXRob3JpdHkwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
AQDZUWf2/BRZ//BYFV14OLHErs04Getf7qYnts7dJ/4WNnjEWIddrZaHF5OuFmBE
RLygBSxfKWDH6Gc5XcpafbxFgoRFnfDqjjOqXCkacZ6YTUjrE5QBYfkGw6577rAo
Ot4na+Fi4hD/dnV2RKSLNZtGwl5yjhdlYqNAl+1TPF9oh0FSjqCEM6L7JSnPsbf+
afx8vbxQxfi8n+XQZ090bHLf0/mdaTZx3kuW579vKMV0XUp56zzxBFQ6to4/pNYY
3uG9WiFbhMhoHClq5Z8mV0nouJ2NFg5hL/y4KMeqPkMlRJizMnVo/BuXdF63kGWJ
Hvfl0ujhTWofHxVS1vhgnqrZAgMBAAGjgagwgaUwHwYDVR0jBBgwFoAU10BlukYi
0n1jHqDIvwut/A0qdHswDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAcYw
HQYDVR0OBBYEFNdAZbpGItJ9Yx6gyL8LrfwNKnR7MEIGCCsGAQUFBwEBBDYwNDAy
BggrBgEFBQcwAYYmaHR0cDovL3Bhdy5zamMucmVkaGF0LmNvbTo5MTgwL2NhL29j
c3AwDQYJKoZIhvcNAQELBQADggEBABHxGsnTSMxDunFBJ8PUN2i3+VHEUPyvsIIH
77QTd95qEA+mejJjhokIf7Nm4MA7WIKAp8IZC5aUIIGWdkf0OG+y1ebcxwdfr+NC
2bAuG0OgT5FCF+YM6AmTSYjZTOAXPrYPdjFiGCo+cx6+soIYiQE6/O7wWFMXlDkx
+3eYvV+kyArH/ABG+dFjBT2qFvRnLU8HpVWJYIOenRQt8GPNW3RAa9EiA5dZVve1
rqD1PiYdTgNAADXGJToeY/X8jpbKnHVF4T0FLE6tjrttI6Tq4dAMOgh6UN0uK2CU
oOzBGyN5UBTmqQHzqq5dXt5H4KIKCCBEJoTjq46VJ0HPvQu5f4AxggE8MIIBOAIB
AzBWMFExHjAcBgNVBAoTFVNqY1JlZGhhdCBEb21haW4gMDEyNDEPMA0GA1UECxMG
cGtpLWNhMR4wHAYDVQQDExVDZXJ0aWZpY2F0ZSBBdXRob3JpdHkCAQYwCQYFKw4D
AhoFAKA+MBcGCSqGSIb3DQEJAzEKBggrBgEFBQcMAjAjBgkqhkiG9w0BCQQxFgQU
OuuE/YrvX8NcpqmrylTJBibBpAAwDQYJKoZIhvcNAQEBBQAEgYBfBvFP7b/LlUZd
FJpqPguPaSgruIo7wLR0rnFki/HacTMg9Eh+b1KnHopeCYTvOIrC10li1F8EEw9J
yBMt+JHfM6P6hOuIo3PF9ciYdEmPBdLHv2itMnU3jeTHfU8qmV1rKkt1VXL47M1Y
HPWxykQ+R28Eet/qPC6dNVqc8AWuqQ==
-----END NEW CERTIFICATE REQUEST-----
```

Chapter 15. CRMFPopClient (Sending an Encoded CRMF Request)

The **CRMFPopClient** utility is a tool to send a Certificate Request Message Format (CRMF) request to a Certificate System CA with the request encoded with proof of possession (POP) data that can be verified by the CA server. If a client provides POP information with a request, the server can verify that the requester possesses the private key for the new certificate.

The tool does all of the following:

1. Has the CA enforce or verify POP information encoded within a CRMF request.
2. Makes simple certificate requests without using the standard Certificate System agent page or interface.
3. Makes a simple certificate request that includes a transport certificate for key archival from the DRM.

NOTE

Key archival is done automatically based on whether a **transport.txt** file (containing the DRM's transport certificate) is in the directory from which the command is run. If the file is present, then the archival is automatically performed; if it is not, then no archival is attempted.
The **transport.txt** must have the entire base 64-encoded transport certificate on a single line with the header and footer removed.

15.1. Syntax

There are two syntax styles for the **CRMFPopClient** utility, depending on the intended use.

This is for sending a simple certificate request to a CA:

CRMFPopClient ***token_password profile_name host port username requester_name pop_option subject_dn*** [OUTPUT_CERT_REQ]

This is for printing the certificate request to stdout, without sending it to a CA:

CRMFPopClient ***token_password pop_option*** OUTPUT_CERT_REQ ***subject_dn***

Option	Description
token_password	The password for the cryptographic token.
profile_name	The CA profile to which to submit the request.
host	The hostname of the CA instance. Depending on how DNS and the network is configured, this can be a machine name, fully-qualified domain name, or IPv4 or IPv6 address.
port	The non-SSL port of the Certificate System CA.
username	The Certificate System user for whom the certificate request is issued.
requester_name	The name of the person or entity who is requesting the certificate.
pop_option	Sets the type of POP request to generate; since

	this can generate invalid requests, this option can be used for testing. There are three values: - **POP_SUCCESS**. Generates a request with the correct POP information; the server verifies that the information is correct. - **POP_FAIL**. Generates a request with incorrect POP information; the server rejects this request if it is submitted. This is used to test server configuration. - **POP_NONE**. Generates a CRMF request with no POP information. If the server is configured to verify all the POP information, then it rejects this request. In that case, it can be used to test the server configuration.
subject_dn	The distinguished name of the requested certificate.
OUTPUT_CERT_REQ	Prints the generated certificate request to the screen. This is optional when the CRMF POP request is sent to a CA, but it is required when the command is used simply to return the request.

15.2. Usage

CRMFPopClient has two different methods of handling the requests it generates: it can send it directly to a CA or it can simply print the request to stdout.

IMPORTANT

A file named **transport.txt** containing the transport certificate in base-64 format *must* be created in the directory from which the utility is launched. This file must be available for archival to a DRM. If the file is present, then the tool picks up this file automatically and performs the key archival.
The **transport.txt** must have the entire base 64-encoded transport certificate on a single line with the header and footer removed.

Sending a request to a CA

The following example generates a CRMF/POP request, has the server verify that the information is correct, and prints the certificate request to the screen:

```
CRMFPopClient secret caUserCert host.example.com 1026 CaUser jsmith POP_SUCCESS
CN=MyTest,C=US,UID=CaUser
```

Printing the request to stdout

A request can simply be printed to stdout. This can be useful if the certificate request is going to be submitted to an external CA or directly through the CA's web services pages. Processing the request, in this case, requires additional manual steps:

1. Use **CRMFPopClient** to generate the request, encoded with POP data.

```
CRMFPopClient secret POP_SUCCESS OUTPUT_CERT_REQ CN=MyTest,C=US,UID=CaUser
```

2. Copy the request that is printed to the screen.

```
MIIFczCCBW8wggTVAgEBMIHygAECpUswSTEaMBgGCgmSJomT8ixkAQETCmptYWdu
ZUNSTUYxCzAJBgNVBAYTAlVTMR4wHAYDVQQDExVqbWFnbmVDUk1GYXJjaGl2ZVRl
c3SmgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJiLbrQaChfzBQLnEnehA3uj
01dA0+pBIJH5PHngjeRpXc6XyYnRpQuFriZUKW7QXewUYQbYsB13F8OwGADfS8wZ
zxfBvLqvQb7h9JtLdsHMVXbQ69/cEs/jCU5Cmr1LmFs4EAAO9Yr/CJjp2hscY82e
KdyGEB6pWuXuBprc8IRJAgMBAAEwggPZMIIDswYJKwYBBQUHBQEEoIIDpDCCA6Ch
FAYIKoZIhvcNAwcECAEBAQEBAQEBgoIBAQBwc6w+H6qZKqQSzQZAOlBc97Uowcjf
YH/vqGsSiN7bkFzx9kEWBZ6hlxP8gY/2JxJQsD0lxsykXcdlC6pW3GwGnBI7obM7
eKeNwL0Mi22ANXdkP7I6KFPFlMNg5v0bynCKOYr2n+ZRQEXnGdLHWnG+vh2GGpDH
1ocXV46dFqeCnSpVEXS/PCcS4I65hByRFMU8lB5vPPBnNJxJt4jY6FU209Y+mrEd
8J2dmtqYLo7y4BhzbBfPn08O1QFJXWGi6ZUbIirZInv4Fg+us1gdIM1wVJSr4rNu
oZx6+JT40ZJ7i0k63T/jMvW77oQesFG21MCOvxrYZJTgTXZ9+sqlKZ/zA4ICgQB6
Dm/JGjAOKdPdpKW1zYs6hpJsjQsLTM5Mz1ONFn7DLe9RDuXdpWOpyjBcqyNqC47Y
CQkRPMW4kj/7XgR4ImycEZZD8OtJF3MqTP7JQGmEXHdsiLRRQy0w/tm0IyI7rJ5p
F34hualY0xtbO+GfaKuUB2GH59Zy11oRug1Okm1UQb/HYuCTL0gh6wH4TXk/g6sx
WVv4cegqsdaZpqAG9+BqvLw9t5R+8dsCCpUTVRg7llEL9HxSAUF2lon9QEEvQJAD
IvofSSXBBf2w+/Qp1x60ZJl7+0vb9P3gEyR3c+BIbIkkdAbfM5knGe2LTnCPcrDb
dY1OV8sgFGxGxcqW2+edJd/yRmsWp/6Dh3HHkd234bUvu+6r5GY7ebueOQIr1HsN
Zwc9XSGLmaShrBTgLyHwq2G3qx7riCCZz6KpSui8YDuQQZE93BoNcuBzvgI/4rIb
uBJfqGYb2t8mSb8Ss+jumbHbZByaVPYp4D9l0Jg3UVbccb19QRIz3G75QotKmDqY
YT7UVbVduLddWN8YvXtoEYcOEfesrdnkEqiHmsALWM0/4U0vWk1Uw7t59O6QMomJ
I8lPc0lZzl1cYaAuuF5SJv/bb/+9S1GqItuult5+bi5t5vN4OE02BfHrpZQHkCbn
ezsIwhDnITwYZSxjMzAeZkBzghTRcNrPwXnvx3crNW2tyZo68FoqOlXAYf/uNBdY
lEBdsvgNPzlRwR63u7pqWA9sJc15X/IwPZ8xj49UwB/cCoSt8PGFADPaAWkSMaT2
rv5+LRkcR56Ol3aMjE9OQEN3kRH75oEGyL5jMkkMa58QGtQgs9WnIhwin0TgWYA2
99wD38RcHVogyQ6Nl4y/MCAGCCsGAQUFBwcXBBTmaclfLv+kkK5z5kTMP54dlnec
UKGBkzANBgkqhkiG9w0BAQQFAAOBgQAqY9mrSqcjPSP9M8p8/TVWdlXn982styAT
DEdau50jksjO/LHPheeFUIaf4+SamE5SUMcEJH9R2p9dqZN8JpvgCYn+h8rjKnIM
5mKstkjtOj42mwizvphkaxIMZdrTSbfC0QjCmkjP2yI3F5QbOoowZ9REH4BMLqRU
sLTu2xgVrw==
```

3. Open the CA's end-entities pages.

```
https://server.example.com:9444/ca/ee/ca/
```

4. Submit the request to any profile which supports CRMF/POP requests.

15.3. Output

When the **CRMFPopClient** command is run successfully, the CA sends back the base 64-encoded version of the certificate request and the HTML page that would have been returned had the request been submitted through the CA web forms.

```
CRMFPopClient password caUserCert test.example.com 9180  joeCRMF joeCRMF
POP_SUCCESS "CN=joeCRMFarchiveTest,C=US, UID=joeCRMF"  OUTPUT_CERT_REQ

Proof Of Possession Utility....

.
.
.
.
.
.
.
.
.
.
.
.
.
.
Generating Legal POP Data.....
.
Signature completed...

Generated Cert Request: ......

MIIFczCCBW8wggTVAgEBMIHygAECpUswSTEaMBgGCgmSJomT8ixkAQETCmptYWdu
ZUNSTUYxCzAJBgNVBAYTAlVTMR4wHAYDVQQDExVqbWFnbmVDUk1GYXJjaGl2ZVRl
c3SmgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJiLbrQaChfzBQLnEnehA3uj
01dA0+pBIJH5PHngjeRpXc6XyYnRpQuFriZUKW7QXewUYQbYsB13F8OwGADfS8wZ
zxfBvLqvQb7h9JtLdsHMVXbQ69/cEs/jCU5Cmr1LmFs4EAAO9Yr/CJjp2hscY82e
KdyGEB6pWuXuBprc8IRJAgMBAAEwggPZMIIDswYJKwYBBQUHBQEEoIIDpDCCA6Ch
FAYIKoZIhvcNAwcECAEBAQEBAQEBgoIBAQBwc6w+H6qZKqQSzQZAOlBc97Uowcjf
YH/vqGsSiN7bkFzx9kEWBZ6hlxP8gY/2JxJQsD0lxsykXcdlC6pW3GwGnBI7obM7
eKeNwL0Mi22ANXdkP7I6KFPFlMNg5v0bynCKOYr2n+ZRQEXnGdLHWnG+vh2GGpDH
1ocXV46dFqeCnSpVEXS/PCcS4I65hByRFMU8lB5vPPBnNJxJt4jY6FU209Y+mrEd
8J2dmtqYLo7y4BhzbBfPn08O1QFJXWGi6ZUbIirZInv4Fg+us1gdIM1wVJSr4rNu
oZx6+JT40ZJ7i0k63T/jMvW77oQesFG21MCOvxrYZJTgTXZ9+sqlKZ/zA4ICgQB6
Dm/JGjAOKdPdpKW1zYs6hpJsjQsLTM5Mz1ONFn7DLe9RDuXdpWOpyjBcqyNqC47Y
CQkRPMW4kj/7XgR4ImycEZZD8OtJF3MqTP7JQGmEXHdsiLRRQy0w/tm0IyI7rJ5p
F34hualY0xtbO+GfaKuUB2GH59Zy11oRug1Okm1UQb/HYuCTL0gh6wH4TXk/g6sx
WVv4cegqsdaZpqAG9+BqvLw9t5R+8dsCCpUTVRg7llEL9HxSAUF2lon9QEEvQJAD
IvofSSXBBf2w+/Qp1x60ZJl7+0vb9P3gEyR3c+BIbIkkdAbfM5knGe2LTnCPcrDb
dY1OV8sgFGxGxcqW2+edJd/yRmsWp/6Dh3HHkd234bUvu+6r5GY7ebueOQIr1HsN
Zwc9XSGLmaShrBTgLyHwq2G3qx7riCCZz6KpSui8YDuQQZE93BoNcuBzvgI/4rIb
uBJfqGYb2t8mSb8Ss+jumbHbZByaVPYp4D9l0Jg3UVbccb19QRIz3G75QotKmDqY
YT7UVbVduLddWN8YvXtoEYcOEfesrdnkEqiHmsALWM0/4U0vWk1Uw7t59O6QMomJ
I8lPc0lZzl1cYaAuuF5SJv/bb/+9S1GqItuult5+bi5t5vN4OE02BfHrpZQHkCbn
ezsIwhDnITwYZSxjMzAeZkBzghTRcNrPwXnvx3crNW2tyZo68FoqOlXAYf/uNBdY
lEBdsvgNPzlRwR63u7pqWA9sJc15X/IwPZ8xj49UwB/cCoSt8PGFADPaAWkSMaT2
rv5+LRkcR56Ol3aMjE9OQEN3kRH75oEGyL5jMkkMa58QGtQgs9WnIhwin0TgWYA2
99wD38RcHVogyQ6Nl4y/MCAGCCsGAQUFBwcXBBTmaclfLv+kkK5z5kTMP54dlnec
UKGBkzANBgkqhkiG9w0BAQQFAAOBgQAqY9mrSqcjPSP9M8p8/TVWdlXn982styAT
DEdau50jksjO/LHPheeFUIaf4+SamE5SUMcEJH9R2p9dqZN8JpvgCYn+h8rjKnIM
5mKstkjtOj42mwizvphkaxIMZdrTSbfC0QjCmkjP2yI3F5QbOoowZ9REH4BMLqRU
sLTu2xgVrw==

End Request:

Server Response.....
--------------------
```

```
<!-- --- BEGIN COPYRIGHT BLOCK ---
     This program is free software; you can redistribute it and/or modify
     it under the terms of the GNU General Public License as published by
     the Free Software Foundation; version 2 of the License.

     This program is distributed in the hope that it will be useful,
     but WITHOUT ANY WARRANTY; without even the implied warranty of
     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
     GNU General Public License for more details.

     You should have received a copy of the GNU General Public License along
     with this program; if not, write to the Free Software Foundation, Inc.,
     51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

     Copyright (C) 2007 Red Hat, Inc.
     All rights reserved.
     --- END COPYRIGHT BLOCK --- -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<script type="text/javascript">
errorReason="Request Deferred - defer request";
requestListSet = new Array;
requestList = new Object;
requestList.requestId="284";
requestListSet[0] = requestList;
errorCode="2";
</script>
<font size="+1" face="PrimaSans BT, Verdana, Arial, Helvetica, sans-serif">
Certificate Profile
</font><br>
  <Font size="-1" face="PrimaSans BT, Verdana, Arial, Helvetica, sans-serif">
<p>
</font>
<table border="0" cellspacing="0" cellpadding="0"
background="/ca/ee/graphics/hr.gif"
width="100%">
  <tr>
    <td> </td>
  </tr>
</table>
<font size="-1" face="PrimaSans BT, Verdana, Arial, Helvetica, sans-serif">
<script language=javascript>

var autoImport = 'false';

if (errorCode == 0) { // processed
  document.write('Congratulations, your request has been processed successfully
');
  document.writeln('<P>');
  for (var i = 0; i < requestListSet.length; i++) {
    document.write('Your request ID is ');
    document.write('<B>'+requestListSet[i].requestId+'</B>.');
    document.writeln('<P>');
  }
  document.writeln('<b>');
  document.writeln('Outputs');
  document.writeln('</b>');
  document.writeln('<P>');
```

```
  document.writeln('<table width=100%>');
for (var i = 0; i < outputListSet.length; i++) {
    document.writeln('<tr valign=top>');
    document.writeln('<td>');
    document.writeln('<FONT size="-1" face="PrimaSans BT, Verdana,
sans-serif">'
);
    document.writeln('<li>');
    document.writeln(outputListSet[i].outputName);
    document.writeln('</FONT>');
    document.writeln('</td>');
    document.writeln('<tr valign=top>');
    document.writeln('</tr>');
    document.writeln('<td>');
    if (outputListSet[i].outputSyntax == 'string') {
      document.writeln(outputListSet[i].outputVal);
    } else if (outputListSet[i].outputSyntax == 'pretty_print') {
      document.writeln('<pre>');
      document.writeln(outputListSet[i].outputVal);
      document.writeln('</pre>');
    }
    document.writeln('</td>');
    document.writeln('</tr>');
}
   document.writeln('</table>');
   document.writeln('<p>');
  document.writeln('<table width=100%>');
    document.writeln('<tr valign=top>');
    document.writeln('<td>');
    document.writeln('<FONT size="-1" face="PrimaSans BT, Verdana,
sans-serif">'
);
    document.writeln('<li>');
    document.writeln('Certificate Imports');
    document.writeln('</FONT>');
    document.writeln('</td>');
   for (var i = 0; i < requestListSet.length; i++) {
    document.writeln('<tr valign=top>');
    document.writeln('<td>');
if (autoImport == 'true') {
    // only support one certificate import
   var loc = "getCertFromRequest?requestId="+ requestListSet[i].requestId +
"&importCert=true";
   document.write("<iframe width='0' height='0' src='"+loc+"' </iframe>");
} else {
    document.writeln('<form method=post action="getCertFromRequest">');
 if (navigator.appName == "Netscape") {
    document.writeln('<input type=hidden name=importCert value=true>');
 } else {
    document.writeln('<input type=hidden name=importCert value=false>');
 }
    document.writeln('<input type=hidden name=requestId value=' +
requestListSet[i].requestId + '>');
    document.writeln('<input type=submit name="Import Certificate"
value="Import Certificate">');
    document.writeln('</form>');
}
    document.writeln('</td>');
    document.writeln('</tr>');
   }
```

```
   document.writeln('</table>');
} else if (errorCode == 1) { // not submitted
  document.write('Sorry, your request is not submitted. The reason is "' +
errorReason + '".');
} else if (errorCode == 2) { // pending
  document.write('Congratulations, your request has been successfully ');
  document.write('submitted. ');
  document.write('Your request will be processed when an authorized agent ');
  document.writeln('verifies and validates the information in your request.');
  document.writeln('<P>');
  for (var i = 0; i < requestListSet.length; i++) {
    document.write('Your request ID is ');
    document.write('<B><a href="checkRequest?requestId=');
    document.write(requestListSet[i].requestId);
    document.write('">'+requestListSet[i].requestId+'</a></B>.');
    document.writeln('<P>');
  }
  document.write('Your can check on the status of your request with ');
  document.write('an authorized agent or local administrator ');
  document.writeln('by referring to this request ID.');
} else if (errorCode == 3) { // rejected
  document.write('Sorry, your request has been rejected. The reason is "' +
errorReason + '"');
  document.writeln('<P>');
  for (var i = 0; i < requestListSet.length; i++) {
    document.write('Your request ID is ');
    document.write('<B>'+requestListSet[i].requestId+'</B>.');
    document.writeln('<P>');
  }
} else { // unknown state
  document.write('Sorry, your request is not submitted. The error code is "' +
errorReason + '".');
}
</script>
</font>
</html>
```

Chapter 16. ExtJoiner (Adding Cerificate Extensions to Requests)

The Certificate System provides policy plug-in modules that allow standard and custom X.509 certificate extensions to be added to end-entity certificates that the server issues. Similarly, the Certificate Setup Wizard that generates certificates for subsystem users allows extensions to be selected and included in the certificates. The wizard interface and the request-approval page of the agent interface contain a text area to paste any extension in its MIME-64 encoded format.

The text field for pasting the extension accepts a single extension blob. To add multiple extensions, they must first be combined into a single extension blob, then pasted into the text field. The **ExtJoiner** tool joins multiple extensions together into a single MIME-64 encoded blob. This new, combined blob can then be pasted in the wizard text field or the request-approval page of the agent interface to specify multiple extensions at once.

16.1. Syntax

The **ExtJoiner** utility has the following syntax:

```
ExtJoiner ext_file0 ext_file1 ... ext_fileN
```

Option	Description
ext_file#	Specifies the path and names for files containing the base-64 DER encoding of an X.509 extension.

16.2. Usage

ExtJoiner does not *generate* an extension in its MIME-64 encoded format; it joins existing MIME-64 encoded extensions. To join multiple custom extensions and add the extensions to a certificate request using **ExtJoiner**, do the following:

1. Find and note the location of the extension program files.
2. Run **ExtJoiner**, specifying the extension files. For example, if there are two extension files named **myExt1** and **myExt2** in a directory called **/etc/extensions**, then the command would be as follows:

   ```
   ExtJoiner /etc/extensions/myExt1 /etc/extensions/myExt2
   ```

   This creates a base-64 encoded blob of the joined extensions, similar to this example:

   ```
   MEwwLgYDVR0lAQHBCQwIgYFKoNFBAMGClGC5EKDM5PeXzUGBi2CVyLNCQYFU
   iBakowGgYDVR0SBBMwEaQPMA0xCzAJBgNVBAYTAlVT
   ```

3. Copy the encoded blob, without any modifications, to a file.
4. Verify that the extensions are joined correctly before adding them to a certificate request by converting the binary data to ASCII using the **AtoB** utility and then dumping the contents of the base-64 encoded blob using the **dumpasn1** utility. For information on the **AtoB** utility, see Chapter 6, *AtoB (Converting ASCII to Binary)*. The **dumpasn1** tool can be downloaded at http://fedoraproject.org/extras/4/i386/repodata/repoview/dumpasn1-0-20050404-1.fc4.html.
   a. Run the **AtoB** utility to convert the ASCII to binary.

```
AtoB  input_file output_file
```

where *input_file* is the path and file containing the base-64 encoded data in ASCII and *output_file* is the path and file for the utility to write the binary output.

b. Run the **dumpasn1** utility.

```
dumpasn1output_file
```

where *output_file* is the path and file containing the binary data. The output looks similar to this:

```
0 30 76: SEQUENCE {
2 30 46: SEQUENCE {
4 06 3: OBJECT IDENTIFIER extKeyUsage (2 5 29 37)
9 01 1: BOOLEAN TRUE
12 04 36: OCTET STRING
 : 30 22 06 05 2A 83 45 04 03 06 0A 51 82 E4 42 83
 : 33 93 DE 5F 35 06 06 2D 82 57 22 CD 09 06 05 51
 : 38 81 6A 4A
 : }
50 30 26: SEQUENCE {
52 06 3: OBJECT IDENTIFIER issuerAltName (2 5 29 18)
57 04 19: OCTET STRING
 : 30 11 A4 0F 30 0D 31 0B 30 09 06 03 55 04 06 13
 : 02 55 53
 : }
 : }

0 warnings, 0 errors.
```

If the output data do not appear to be correct, check that the original Java™ extension files are correct, and repeat converting the files from ASCII to binary and dumping the data until the correct output is returned.

5. When the extensions have been verified, copy the base-64 encoded blob that was created by running **ExtJoiner** to the Certificate System wizard screen, and generate the certificate or the certificate signing request (CSR).

Chapter 17. GenExtKeyUsage (Adding the Key Usage Extension to a Request)

The **GenExtKeyUsage** tool creates a base-64 encoded blob that adds **ExtendedKeyUsage** (OID 2.5.29.37) to the certificate. This blob is pasted into the certificate approval page when the certificate is created.

17.1. Syntax

The **GenExtKeyUsage** tool has the following syntax:

```
GenExtKeyUsage [true|false] OID ...
```

Option	Description
true | false	Sets the criticality. **true** means the extension is critical; **false** means it is not critical. The criticality value is used during the certificate validation process. If an extension is marked as critical, then the path validation software must be capable of interpreting that extension.
OID	The OID numbers that represent each certificate type selected for the certificate.

For more information on the OIDs that can be used for each certificate type, refer to the "Extended Key Usage Extension Default" section in appendix B, "Certificate and CRL Extensions," in the *Certificate System Administrator's Guide*.

Chapter 18. GenIssuerAltNameExt (Adding the Issuer Name Extention to a Request)

The **GenIssuerAltNameExt** creates a base-64 encoded blob that adds the issuer name extensions, **IssuerAltNameExt** (OID 2.5.29.18), to the new certificate. This blob is pasted into the certificate approval page when the certificate is created.

18.1. Syntax

The **GenIssuerAltNameExt** tool uses parameter pairs where the first parameter specifies the general type of name attribute which is used for the issuer and the second parameter gives that name in that format. The tool has the following syntax:

GenIssuerAltNameExt ***general_type#*** ... ***general_name#*** ...

Parameter	Description
general_type	Sets the type of name. It can be one of the following strings: - RFC822Name - DNSName - EDIPartyName - URIName - IPAddressName - OIDName - X500Name
general_name	A string, conforming to the name type, that gives the name of the issuer. - For **RFC822Name**, the value must be a valid Internet mail address. For example, *testCA@example.com*. - For **DNSName**, the value must be a valid fully-qualified domain name. For example, *testCA.example.com*. - For **EDIPartyName**, the value must be an IA5String. For example, *Example Corporation*. - For **URIName**, the value must be a non-relative URI following the URL syntax and encoding rules. The name must include both a scheme, such as **http**, and a fully qualified domain name or IP address of the host. For example, *http://testCA.example.com*. - For **IPAddressName**, the value must be a valid IP address. An IPv4 address must be in the format *n.n.n.n* or *n.n.n.n,m.m.m.m*. For example, *128.21.39.40* or *128.21.39.40,255.255.255.00*. An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example,

0:0:0:0:0:0:13.1.68.3, *FF01::43*, *0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*.

- For **`OIDName`**, the value must be a unique, valid OID specified in dot-separated numeric component notation. For example, *1.2.3.4.55.6.5.99*.
- For **`X500Name`**, the value must be a string form of X.500 name, similar to the subject name in a certificate. For example, *cn=SubCA, ou=Research Dept, o=Example Corporation, c=US*.

18.2. Usage

The following example sets the issuer name in the **`RFC822Name`** and **`X500Name`** formats:

```
GenIssuerAltNameExt RFC822Name TomTom@example.com X500Name cn=TomTom
```

Chapter 19. SubjectAltNameExt (Adding the Subject Alternative Name Extension to a Request)

The **GenSubjectAltNameExt** creates a base-64 encoded blob to add the alternate subject name extension, **SubjectAltNameExt** (OID 2.5.29.17), to the new certificate. This blob is pasted into the certificate approval page when the certificate is created.

19.1. Syntax

The **GenSubjectAltNameExt** tool uses parameter pairs where the first parameter specifies the type of name format, and the second parameter gives that name in the specified format.

This tool has the following syntax:

GenSubjectAltNameExt ***general_type#*** ... ***general_name#*** ...

Parameter	Description
general_type	Sets the type of name that is used. This can be any of the following strings: - **RFC822Name** - **DNSName** - **EDIPartyName** - **URIName** - **IPAddressName** - **OIDName** - **X500Name**
general_name	A string, conforming to the specified format, of the subject name. - For **RFC822Name**, the value must be a valid Internet mail address. For example, **testCA@example.com**. - For **DNSName**, the value must be a valid fully-qualified domain name. For example, **testCA.example.com**. - For **EDIPartyName**, the value must be an IA5String. For example, **Example Corporation**. - For **URIName**, the value must be a non-relative URI following the URL syntax and encoding rules. The name must include both a scheme, such as **http**, and a fully qualified domain name or IP address of the host. For example, **http://testCA.example.com**. - For **IPAddressName**, the value must be a valid IP address. An IPv4 address must be in the format **n.n.n.n** or **n.n.n.n,m.m.m.m**. For example, **128.21.39.40** or **128.21.39.40,255.255.255.00**. An IPv6 address uses a 128-bit namespace, with the

IPv6 address separated by colons and the netmask separated by periods. For example, **0:0:0:0:0:0:13.1.68.3**, **FF01::43**, **0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0**, and **FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000**.

- For **OIDName**, the value must be a unique, valid OID specified in dot-separated numeric component notation. For example, **1.2.3.4.55.6.5.99**.
- For **X500Name**, the value must be a string form of X.500 name, similar to the subject name in a certificate. For example, **cn=SubCA, ou=Research Dept, o=Example Corporation, c=US**.

19.2. Usage

In the following example, the subject alternate names are set to the **RFC822Name** and **X500Name** types.

```
GenSubjectAltNameExt RFC822Name TomTom@example.com X500Name cn=TomTom
```

Chapter 20. HttpClient (Sends a Request over HTTP)

The HTTP Client utility, **HttpClient**, sends a CMC request (created with the CMC Request utility) or a PKCS #10 request to a CA.

20.1. Syntax

This utility takes a single **.cfg** configuration file as a parameter. The syntax is as follows:

HttpClient ***/path/to/file.cfg***

The **.cfg** file has the following parameters:

Parameters	Description
host	The hostname for the Certificate System server. Depending on how DNS and the network are configured, this can be a machine name, fully-qualified domain name, or IPv4 or IPv6 address. For example, **host=server.example.com**.
port	Any port number for Certificate System server. For example, **port=9443**.
secure	**true** for an HTTPS connection, **false** for an HTTP connection. For example, **secure=true**.
input	The full path and filename for the enrollment request, which must be in binary format. For example, **input=cmcReqCRMFBin**.
output	The full path and filename for the response in binary format. For example, **output=cmcResp**.
dbdir	The full path to the directory where the **cert8.db**, **key3.db**, and **secmod.db** databases are located. This parameter is ignored if **secure=false**. For example, **dbdir=/usr/bin**.
clientmode	**true** for client authentication, **false** for no client authentication. This parameter is ignored if **secure=false**. For example, **clientmode=true**.
password	The password for the **cert8.db** database. This parameter is ignored if **secure=false** and **clientauth=false**. For example, **password=secret**.
nickname	The nickname of the client certificate. This parameter is ignored if **clientmode=false**. For example, **nickname=CS Agent-102504a's 102504a ID**.
servlet	The URI of the servlet that processes full CMC requests. The default value is **/ca/profileSubmitCMCFull**. For example, **servlet=/ca/profileSubmitCMCFull**.

Chapter 21. OCSPClient (Sending an OCSP Request)

The OCSP request utility, **OCSPClient**, creates an OCSP request conforming to RFC 2560, submits it to the OCSP server, and saves the OCSP response in a file.

21.1. Syntax

The **OCSPClient** tool has the following syntax:

OCSPClient ***host port dbdir nickname serial_number*** or ***filename output times***

Option	Description
host	Specifies the hostname of the OCSP server. Depending on how DNS and the network are configured, this can be a machine name, fully-qualified domain name, or IPv4 or IPv6 address.
port	Gives the end-user port number of the OCSP server.
dbdir	Gives the location of the security databases (**cert8.db**, **key3.db**, and **secmod.db**) which contain the CA certificate that signed the certificate being checked.
nickname	Gives the CA certificate nickname.
serial_number or *filename*	Gives the serial number or, alternatively, the name of the file containing the request for the certificate that's status is being checked.
output	Gives the path and file to which to print the DER-encoded OCSP response.
times	Specifies the number of times to submit the request.

Chapter 22. PKCS10Client (Generating a PKCS #10 Certificate Request)

The PKCS #10 utility, **PKCS10Client**, generates a 1024-bit RSA key pair in the security database, constructs a PKCS#10 certificate request with the public key, and outputs the request to a file.

PKCS #10 is a certification request syntax standard defined by RSA. A CA may support multiple types of certificate requests. The Certificate System CA supports KEYGEN, PKCS#10, CRMF, and CMC.

To get a certificate from the CA, the certificate request needs to be submitted to and approved by a CA agent. Once approved, a certificate is created for the request, and certificate attributes, such as extensions, are populated according to certificate profiles.

22.1. Syntax

The **PKCS10Client** tool has the following syntax:

```
PKCS10Client -p certDBPassword -d certDBDirectory -o outputFile -s subjectDN
```

Option	Description
p	Gives the password for the security databases.
d	Gives the path to the security databases.
o	Sets the path and filename to output the new PKCS #10 certificate in base 64 format.
s	Gives the subject DN of the certificate.

Chapter 23. PKCS12Export (Exports Certificates and Keys from a Database)

The **PKCS12Export** simply dumps all of the certificates and corresponding keys in an NSS security database to a specified **.p12** output file.

23.1. Syntax

PKCS12Export -d ***/path/to/cert-directory*** -p ***keydb-password-file*** -w ***pkcs12-password-file*** -o ***output-file.p12*** [-debug]

Parameters	Description
-d	Gives the full path to the NSS databases which contain the certificate to export.
-o	Gives the name of the file to output the exported certificate to.
-p	Gives the full path and filename of a file containing the password to access the NSS security databases.
-w	Gives the full path and filename of a file to use to set the password to access the output file.
-debug	Turns on debugging output to stdout.

23.2. Usage and Output

The **PKCS12Export** command exports each certificate in the database into the **.p12** output file. When the **-debug** option is used, the certificate nickname for each certificate is printed to stdout as the operation proceeds. (Otherwise, there is no output from the command.)

```
# PKCS12Export -debug -d /var/lib/pki-ca/alias -p dbpwd.txt -w p12pwd.txt -o
master.p12
PKCS12Export debug: The directory for certdb/keydb is .
PKCS12Export debug: The password file for keydb is dbpwd.txt
PKCS12Export debug: Number of user certificates = 5
PKCS12Export debug: Certificate nickname = ocspSigningCert cert-ca
PKCS12Export debug: Private key is not null
PKCS12Export debug: Certificate nickname = subsystemCert cert-ca
PKCS12Export debug: Private key is not null
PKCS12Export debug: Certificate nickname = caSigningCert cert-ca
PKCS12Export debug: Private key is not null
PKCS12Export debug: Certificate nickname = Server-Cert cert-ca
PKCS12Export debug: Private key is not null
PKCS12Export debug: Certificate nickname = auditSigningCert cert-ca
PKCS12Export debug: Private key is not null
```

Chapter 24. revoker (Sending Revocation Requests)

The **revoker** utility sends revocation requests to the CA agent interface to revoke certificates. To access the interface, **revoker** needs to have access to an agent certificate that is part of the subsystem group that is acceptable to the CA.

The **revoker** tool can do all of the following:

- Specify which certificate or a list of certificates to revoke by listing the hexadecimal serial numbers.
- Specify a revocation reason.
- Specify an invalidity date.
- Unrevoke a certificate that is currently on hold.

24.1. Syntax

The **revoker** utility has the following syntax:

revoker -s ***serialNumber*** -n ***rsa_nickname*** [[-p ***password***] | [-w ***passwordFile***]] [-d ***dbdir***] [-v] [-V] [-u] [-r ***reasoncode***] [-i ***numberOfHours***] hostname [:port]

Option	Description
s	Gives the serial numbers in hexadecimal of the certificates to revoke. A hexadecimal serial number, for example, is like **0x31**, or multiple serial numbers can be listed separated by commas, such as **0x44,0x64,0x22**.
n	Gives the agent certificate nickname.
p	Gives the certificate database password. Not used if the **-w** option is used.
w	*Optional.* Gives the path to the password file. Not used if the **-p** option is used.
d	*Optional.* Gives the path to the security databases.
v	*Optional.* Sets the operation in verbose mode.
V	*Optional.* Gives the version of the **revoker** tool.
u	*Optional.* Unrevokes a certificate, meaning that certificate status is changed from on hold to active.
r	Gives the reason to revoke the certificate. The following are the possible reasons: - *0* - Unspecified (default). - *1* - The key was compromised. - *2* - The CA key was compromised. - *3* - The affiliation of the user has changed. - *4* - The certificate has been superseded. - *5* - Cessation of operation. - *6* - The certificate is on hold.
i	Sets the invalidity date in hours from current time for when to revoke the certificate.

hostname	Gives the hostname of the server to which to send the request. Depending on how DNS and the network are configured, this can be a machine name, fully-qualified domain name, or IPv4 or IPv6 address.
port	*Optional.* Gives the agent's SSL port number of the server.

24.2. Output

Without using the verbose option (**-v**), **revoker** returns an exit code of 0, without any additional output the standard I/O.

With the **-v** option, the command shows the GET request sent to the CA agent interface and then the results (in an HTML page) that is returned.

```
# revoker -d . -s 0x17 -n "CA Administrator of Instance pki-ca Example Domain" -p
secret -v -r 6 -i 1 server.example.com:9443


GET /ca/doRevoke?
op=doRevoke&revocationReason=6&invalidityDate=1299187797000&revokeAll=(|(certRecor
dId%3D0x17))&totalRecordCount=1 HTTP/1.0
port: 9443
addr='server.example.com'
family='2'
Subject: CN=server.example.com,OU=pki-ca,O=Example Domain
Issuer : CN=Certificate Authority,OU=pki-ca,O=Example Domain
-- SSL3: Server Certificate Validated.
Called mygetclientauthdata - nickname = CA Administrator of Instance pki-ca
Example Domain ID
   mygetclientauthdata - cert = 8da87b8
   mygetclientauthdata - privkey = 8de65a8
PR_Write wrote 143 bytes from bigBuf
bytes: [GET /ca/doRevoke?
op=doRevoke&revocationReason=6&invalidityDate=1299187797000&revokeAll=(|(certRecor
dId%3D0x17))&totalRecordCount=1 HTTP/1.0

]
do_writes shutting down send socket
do_writes exiting with (failure = 0)
bulk cipher RC4, 128 secret key bits, 128 key bits, status: 1
connection 1 read 9000 bytes (9000 total).
these bytes read:
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html
Date: Thu, 03 Mar 2011 22:29:58 GMT
Connection: close

<!-- --- BEGIN COPYRIGHT BLOCK ---
     This program is free software; you can redistribute it and/or modify
     it under the terms of the GNU General Public License as published by
     the Free Software Foundation; version 2 of the License.

     This program is distributed in the hope that it will be useful,
     but WITHOUT ANY WARRANTY; without even the implied warranty of
     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
     GNU General Public License for more details.

     You should have received a copy of the GNU General Public License along
     with this program; if not, write to the Free Software Foundation, Inc.,
     51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

     Copyright (C) 2007 Red Hat, Inc.
     All rights reserved.
     --- END COPYRIGHT BLOCK --- -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE>Revocation Result</TITLE>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<SCRIPT LANGUAGE="JavaScript">
var header = new Object();
```

```
var fixed = new Object();
var recordSet = new Array;
var result = new Object();
var httpParamsCount = 0;
var httpHeadersCount = 0;
var authTokenCount = 0;
var serverAttrsCount = 0;
header.HTTP_PARAMS = new Array;
header.HTTP_HEADERS = new Array;
header.AUTH_TOKEN = new Array;
header.SERVER_ATTRS = new Array;
header.dirEnabled = "no";
header.error = null;
header.revoked = "yes";
header.totalRecordCount = 1;
var recordCount = 0;
var record;
record = new Object;
record.HTTP_PARAMS = new Array;
record.HTTP_HEADERS = new Array;
record.AUTH_TOKEN = new Array;
record.SERVER_ATTRS = new Array;
record.error=null;
record.serialNumber="17";
recordSet[recordCount++] = record;
record.recordSet = recordSet;
result.header = header;
result.fixed = fixed;
result.recordSet = recordSet;
</SCRIPT>

<BODY bgcolor="white">
<SCRIPT type="text/javascript">
//<!--
function toHex1(number)
{
    var absValue = "", sign = "";
    var digits = "0123456789abcdef";
    if (number < 0) {
        sign = "-";
        number = -number;
    }

    for(; number >= 16 ; number = Math.floor(number/16)) {
        absValue = digits.charAt(number % 16) + absValue;
    }
    absValue = digits.charAt(number % 16) + absValue;
    return sign + '0x' + absValue;
}

function toHex(number)
{
    return '0x' + number;
}

if (result.header.revoked == 'yes') {
    document.write('<font size="+1" face="PrimaSans BT, Verdana, Arial, Helvetica,
sans-serif">');
    document.writeln('Certificate Revocation Has Been Completed</font><br><br>');
    if (result.recordSet.length == 0 && result.header.totalRecordCount > 0) {
```

```
        document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
        document.write('All requested certificates were already revoked.');
        document.writeln('</font><br>');
    } else if (result.recordSet.length == 1) {
        if (result.recordSet[0].error == null) {
           document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
            document.writeln('Certificate with serial number <b>' +
                             toHex(result.recordSet[0].serialNumber) +
                             '</b> has been revoked.');
            document.writeln('</font><br>');

            document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
            if (result.header.updateCRL && result.header.updateCRL == "yes") {
                if (result.header.updateCRLSuccess != null &&
                    result.header.updateCRLSuccess == "yes") {
                    document.writeln('The Certificate Revocation List has been
successfully updated.');
                } else {
                    document.writeln('The Certificate Revocation List update
Failed');
                    if (result.header.updateCRLSuccess != null)
                        document.writeln(' with error '+
result.header.updateCRLError);
                    else
                        document.writeln('. No further details provided.');
                }
            } else {
                document.writeln(
                    'The Certificate Revocation List will be updated '+
                    'automatically at the next scheduled update.');
            }
            document.writeln('</font><br>');
/*
            if (result.header.dirEnabled != null && result.header.dirEnabled ==
'yes') {
                document.writeln('<font size="-1" face="PrimaSans BT, Verdana,
Arial, Helvetica, sans-serif">');
                if (result.header.certsUpdated > 0) {
                    document.write('Directory has been successfully updated.');
                } else {
                    document.write('Directory has not been updated.  See log files
for more details.');
                }
                document.writeln('</font><br>');
            }
*/
        } else {
           document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
            document.writeln('Certificate with serial number <b>' +
                             toHex(result.recordSet[0].serialNumber) +
                             '</b> is not revoked.<br><br>');
            document.writeln('Additional Information:');
            document.writeln('</font>');
            document.writeln('<blockquote>');
           document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
```

```
            document.writeln(result.recordSet[0].error);
            document.writeln('</font>');
            document.writeln('</blockquote>');
        }
    } else if (result.recordSet.length > 1) {
        document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
        document.write('The following certificates were processed to complete
revocation request:');
        document.writeln('</font>');

        document.writeln('<blockquote>');
        document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
        var revokedCerts = 0;
        for(var i = 0; i < result.recordSet.length; i++) {
            if (result.recordSet[i].error == null) {
                revokedCerts++;
                document.writeln(toHex(result.recordSet[i].serialNumber) + ' -
revoked<BR>\n');
            } else {
                document.write(toHex(result.recordSet[i].serialNumber) + ' -
failed');
                if (result.recordSet[i].error != null)
                    document.write(': ' + result.recordSet[i].error);
                document.writeln('<BR>\n');
            }
        }
        document.writeln('</font>');
        document.write('</blockquote>');

        if (revokedCerts > 0 && result.header.dirEnabled != null &&
result.header.dirEnabled == 'yes') {
            document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
            if (result.header.updateCRL && result.header.updateCRL == "yes") {
                if (result.header.updateCRLSuccess != null &&
                    result.header.updateCRLSuccess == "yes") {
                    document.writeln('The Certificate Revocation List has been
successfully updated.');
                } else {
                    document.writeln('The Certificate Revocation List update
Failed');
                    if (result.header.updateCRLSuccess != null)
                        document.writeln(' with error '+
                                         result.header.updateCRLError);
                    else
                        document.writeln('. No further details provided.');
                }
            } else {
                document.writeln(
                    'The Certificate Revocation List will be updated '+
                    'automatically at the next scheduled update.');
            }
            document.writeln('<br>');
/*
            if (result.header.certsUpdated > 0) {
                if (result.header.certsUpdated == result.header.certsToUpdate) {
                    document.write('Directory has been successfully updated.');
                } else {
```

```
                    document.write('Directory has been partially updated.  See log
files for more details.');
                }
            } else {
                document.write('Directory has not been updated.  See log files for
more details.');
            }
*/
            document.writeln('</font><br>');
        }
    }
} else if (result.header.revoked == 'pending') {
    document.write('<font size="+1" face="PrimaSans BT, Verdana, Arial, Helvetica,
sans-serif">');
 document.writeln('Revocation Request Has Been Submitted</font><br><br>');
} else if (result.header.revoked == 'rejected') {
    document.write('<font size="+1" face="PrimaSans
connection 1 read 1249 bytes (10249 total).
these bytes read:
 BT, Verdana, Arial, Helvetica, sans-serif">');
    document.writeln('Certificate Revocation Has Been Rejected</font><br><br>');
    if (result.header.error != null) {
        document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">Additional information:</font>');
        document.writeln('<blockquote>');
     document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
        document.writeln(result.header.error);
        document.writeln('</font>');
        document.writeln('</blockquote>');
    }
} else {
    document.write('<font size="+1" face="PrimaSans BT, Verdana, Arial, Helvetica,
sans-serif">');
 document.writeln('Revocation Request Cannot Be Completed</font><br><br>');
    if (result.header.error != null) {
        document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">Additional information:</font>');
        document.writeln('<blockquote>');
     document.writeln('<font size="-1" face="PrimaSans BT, Verdana, Arial,
Helvetica, sans-serif">');
        document.writeln(result.header.error);
        document.writeln('</font>');
        document.writeln('</blockquote>');
    }
}
//-->
</SCRIPT>
</BODY>
</HTML>

connection 1 read 10249 bytes total. -----------------------------
```

Chapter 25. tpsclient (Debugging the TPS)

The **tpsclient** tool can be used for debugging or testing the TPS. The **tpsclient** imitates the Enterprise Security Client and can give debug output or emulate enrolling and formatting tokens without having to use tokens.

The **tpsclient** tool is launched by running the command **tpsclient**. The tool has no options. Running this opens a shell which allows specific commands to be directed toward the **tpsclient**.

```
tpsclient

Registration Authority Client
'op=help' for Help
Command>
```

tpsclient and the TPS need to agree on a set of symmetric keys to establish a secure channel. They are both configured with a mutual default token, which has the default key set (**version 1**) which contains three keys: authentication key, Mac key, and key encryption key (KEK). The TPS subsystem understands and accepts the default key set.

The default key values for each are set to **0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f**, 16 bytes. The default configuration is shown by running the **token_status** option within the **tpsclient** command shell.

```
Command>token_status
token_status
Output> life_cycle_state : '0'
Output> pin : 'password'
Output> app_ver : '00010203' (4 bytes)
Output> major_ver : '0'
Output> minor_ver : '0'
Output> cuid : '00010203040506070809' (10 bytes)
Output> msn : '00000000' (4 bytes)
Output> key_info : '0101' (2 bytes)
Output> auth_key : '404142434445464748494a4b4c4d4e4f' (16 bytes)
Output> mac_key : '404142434445464748494a4b4c4d4e4f' (16 bytes)
Output> kek_key : '404142434445464748494a4b4c4d4e4f' (16 bytes)
Result> Success - Operation 'token_status' Success (8 msec)
Command>
```

If the TPS is configured to use a new master key, then the **tpsclient** must also be reconfigured, or it cannot establish its connection to the TPS.

1. Get the new key set data to input into **tpsclient**. The default key set must be stored in the TKS, and the master key must be added. Do this by editing the TKS mapping parameter in the TKS **CS.cfg** file:

   ```
   tks.mk_mappings.#02#01=nethsm1:masterkey
   ```

   This configuration instructs the TKS to map the master key named **masterkey** on the **nethsm1** token to the **#02#01** key.
2. Enable key upgrade in the TPS by editing the update symmetric keys parameter in the TPS **CS.cfg** file:

```
op.format.tokenKey.update.symmetricKeys.enable=true
op.format.tokenKey.update.symmetricKeys.requiredVersion=2
```

This setting instructs the TPS to upgrade the token from version 1 to version 2 during the **tpsclient** format operation.

3. Format the token using **tpsclient**, as follows:

```
tpsclient
Command>op=token_set cuid=a00192030405060708c9 app_ver=6FBBC105
key_info=0101
Command>op=token_set auth_key=40414243444546474849 4a4b4c4d4e4f
Command>op=token_set mac_key=40414243444546474849 4a4b4c4d4e4f
Command>op=token_set kek_key=40414243444546474849 4a4b4c4d4e4f

Command>op=ra_format uid=jsmith pwd=password num_threads=1 new_pin=password
```

The CUID can be any 10-byte string; it affects how the TKS computes the new key set for **tpsclient**.

TIP

Because it can be tedious to type each operation and parameter through the command line, it is possible to create an input file and then point the **tpsclient** command to the file. For example:

```
tpsclient < /tmp/input.txt
```

Example 25.1, “Example tpsclient Enrollment Input File” and Example 25.2, “Example tpsclient Format Input File” both list examples for an input file.
The command prompt will return any output given by **tpsclient** during the operation and the final result of the command.

4. After running the format operation, **tpsclient** prints the new key set in the standard output. Save the new values in a new **tpsclient** input file. The input file can then be used with a production TPS server.

tpsclient can be used for formatting operations or for enrollment operations. The sample input file for an enrollment operation is shown in Example 25.1, “Example tpsclient Enrollment Input File”.

Example 25.1. Example tpsclient Enrollment Input File

```
op=var_set name=ra_host value=server.example.com
op=var_set name=ra_port value=7888
op=var_set name=ra_uri value=/nk_service
op=token_set cuid=00000000000000000001
  msn=01020304 app_ver=6FBBC105 key_info=0101 major_ver=0 minor_ver=0
op=token_set auth_key=40414243444546474849 4a4b4c4d4e4f
op=token_set mac_key=40414243444546474849 4a4b4c4d4e4f
op=token_set kek_key=40414243444546474849 4a4b4c4d4e4f

op=ra_enroll uid=jdoe pwd=password new_pin=password num_threads=1
```

The sample input file for an enrollment operation is shown in Example 25.2, “Example tpsclient Format Input File”.

Example 25.2. Example tpsclient Format Input File

```
op=var_set name=ra_host value=server.example.com
op=var_set name=ra_port value=7888
op=var_set name=ra_uri value=/nk_service
op=token_set cuid=00000000000000000001
  msn=01020304 app_ver=6FBBC105 key_info=0101 major_ver=0 minor_ver=0
op=token_set auth_key=40414243444546474849 4a4b4c4d4e4f
op=token_set mac_key=40414243444546474849 4a4b4c4d4e4f
op=token_set kek_key=40414243444546474849 4a4b4c4d4e4f
op=ra_format uid=jsmith pwd=secret new_pin=newsecret num_threads=1
```

NOTE

The host value can be an IPv4 address or an IPv6 address, if one is configured for the host.

25.1. Syntax

The **tpsclient** tool has the following syntax:

```
tpsclient op=operation options
```

Table 25.1. tpsclient Operations

Operation	Description	Options
op=help	Brings up the help page, which lists all usage and options for the **tpsclient** tool.	N/A
op=debug filename=*filename*	Enables debugging.	*filename* sets the debug file.
op=ra_enroll	Tests certificate enrollments.	• **uid** gives the user ID of the user running. • **pwd** gives the password corresponding to the user ID. • **num_threads** sets the number of threads to use • **secureid_pin** gives the token password • **keygen** set whether server-side key generation is enabled.
op=ra_reset_pin	Resets the token PIN.	• **uid** gives the user ID of the user running. • **pwd** gives the password corresponding to the user ID. • **num_threads** sets the number of threads to use • **secureid_pin** gives the token password • **new_pin** sets the new PIN (token password).
op=ra_update	Updates the applet.	• **uid** gives the user ID of the user running. • **pwd** gives the password corresponding to the user ID. • **num_threads** sets the number of threads to use • **secureid_pin** gives the token password
op=token_set	Sets the token value.	The usage with this operation is *name=value*, which sets the token name and description.
op=token_status	Returns the current token status/	N/A
op=var_get	Gets the current value of the variable.	This has the usage **name=***name*, where *name* is the variable being checked.
op=var_list	Lists all possible variables.	N/A

op=var_set	Sets variable values.	- **name** sets the name of the variable. - **value** sets the value of the named variable.

Chapter 26. DRMTool (Rewrapping Private Keys)

Some private keys (mainly in older deployments) were wrapped in SHA-1, 1024-bit storage keys when they were archived in the DRM. These algorithms have become less secure as processor speeds improve and algorithms have been broken. As a security measure, it is possible to rewrap the private keys in a new, stronger storage key (SHA-256, 2048-bit keys).

NOTE

Because the **DRMTool** utility can export private keys from one DRM, rewrap them with a new storage key, and then import them into a new DRM, this tool can be used as part of a process of combining multiple DRM instances into a single DRM.

26.1. Syntax

The **DRMTool** utility can be run to rewrap keys, to renumber keys, or both.

The syntax for rewrapping keys:

```
DRMTool -drmtool_config_file /path/to/tool_config_file
 -source_ldif_file /path/to/original_ldif_file
 -target_ldif_file /path/to/newinstance_ldif_file
 -log_file /path/to/tool_log_file
 [-source_pki_security_database_path /path/to/nss_databases
 -source_storage_token_name /path/to/token
 -source_storage_certificate_nickname storage_certificate_nickname
 -target_storage_certificate_file /path/to/new_ASCII_storage_cert
 [-source_pki_security_database_pwdfile /path/to/password_file]]
 [-source_drm_naming_context name -target_drm_naming_context name]
 [-process_requests_and_key_records_only]
```

The syntax to renumber keys:

```
DRMTool -drmtool_config_file /path/to/tool_config_file
 -source_ldif_file /path/to/original_ldif_file
 -target_ldif_file /path/to/newinstance_ldif_file
 -log_file /path/to/tool_log_file
 [-append_id_offset prefix_to_add | -remove_id_offset prefix_to_remove]
 [-source_drm_naming_context name -target_drm_naming_context name]
 [-process_requests_and_key_records_only]
```

Option	Description
Mandatory parameters	
-drmtool_config_file	Gives the complete path and filename of the confguration file used by the tool. This configuration process tells the tool how to process certain parameters in the existing key records, whether to apply any formatting changes (like changing the naming context or adding an offset) or even whether to update the modify date. The configuration file is required and a default file is included with the tool. The file format is

	described in Section 26.2, ".cfg File".
-source_ldif_file	Gives the complete path and filename of the LDIF file which contains all of the key data from the *old* DRM.
-target_ldif_file	Gives the complete path and filename of the LDIF file to which the tool will write all of the key data from the *new* DRM. This file is created by the tool as it runs.
-log_file	Gives the path and filename of the log file to use to log the tool progress and messages. This file is created by the tool as it runs.
Optional parameters	
-source_drm_naming_context	Gives the naming context of the original DRM instance, the DN element that refers to the original DRM. Key-related LDIF entries have a DN with the DRM instance name in it, such as *cn=1,ou=kra,ou=requests,dc=alpha.example.com-pki-kra*. The naming context for that entry is the DN value, *alpha.example.com-pki-kra*. These entries can be renamed, automatically, from the old DRM instance naming context to the new DRM instance naming context. While this argument is optional, it is recommended because it means that the LDIF file does not have to be edited before it is imported into the target DRM. If this argument is used, then the **-target_drm_naming_context** argument must also be used.
-target_drm_naming_context	Gives the naming context of the new DRM instance, the name that the original key entries should be changed *to*. Key-related LDIF entries have a DN with the DRM instance name in it, such as *cn=1,ou=kra,ou=requests,dc=omega.example.com-pki-kra*. The naming context for that entry is the DN value, *omega.example.com-pki-kra*.These entries can be renamed, automatically, from the old DRM instance to the new DRM instance naming context. While this argument is optional, it is recommended because it means that the LDIF file does not have to be edited before it is imported into the target DRM. If this argument is used, then the **-source_drm_naming_context** argument must also be used.
-process_requests_and_key_records_only	Removes configuration entries from the source

	LDIF file, leaving only the key and request entries. While this argument is optional, it is recommended because it means that the LDIF file does not have to be edited before it is imported into the target DRM.
Rewrap parameters	
-source_pki_security_database_path	Gives the full path to the directory which contains the NSS security databases used by the *old* DRM instance. This option is required if any other rewrap parameters are used.
-source_storage_token_name	Gives the name of the token which stores the DRM data, like *Internal Key Storage Token* for internal tokens or a name like *NHSM6000-OCS* for the hardware token name. This option is required if any other rewrap parameters are used.
-source_storage_certificate_nickname	Gives the nickname of the DRM storage certificate for the *old* DRM instance. Either this certificate will be located in the security database for the *old* DRM instance or the security database will contain a pointer to the certificate in the hardware token. This option is required if any other rewrap parameters are used.
-target_storage_certificate_file	Gives the path and filename of an ASCII-formatted file of the storage certificate for the *new* DRM instance. The storage certificate should be exported from the new DRM's databases and stored in an accessible location before running **DRMTool**. This option is required if any other rewrap parameters are used.
-source_pki_security_database_pwdfile	Gives the path and filename to a password file that contains only the password for the storage token given in the **-source_storage_token_name** option. This argument is optional when other rewrap parameters are used. If this argument is not used, then the script prompts for the password.
Number offset parameters	
-append_id_offset	Gives an ID number which will be preprended to every imported key, to prevent possible collisions. A unique ID offset should be used for every DRM instance which has keys exported using **DRMTool**.

	If **-append_id_offset** is used, then **do not** use the **-remove_id_offset** option.
-remove_id_offset	Gives an ID number to remove from the beginning of every imported key. If **-remove_id_offset** is used, then **do not** use the **-append_id_offset** option.

26.2. .cfg File

The required configuration file instructs the **DRMTool** how to process attributes in the key archival and key request entries in the LDIF file. There are six types of entries:

- CA enrollment requests
- TPS enrollment requests
- CA key records
- TPS key records
- CA and TPS recovery requests (which are treated the same in the DRM)

Each key and key request has an LDAP entry with attributes that are specific to that kind of record. For example, for a recovery request:

```
dn: cn=1,ou=kra,ou=requests,dc=alpha.example.com-pki-kra
objectClass: top
objectClass: request
objectClass: extensibleObject
requestId: 011
requestState: complete
dateOfCreate: 20110121181006Z
dateOfModify: 20110524094652Z
extdata-drm--005ftrans--005fdeskey: 3#C7#82#0F#5D#97GqY#0Aib#966#E5B#F56#F24n#
 F#9E#98#B3
extdata-public--005fkey: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDu6E3uG+Ep27bF1
 yTWvwIDAQAB
extdata-archive: true
extdata-requesttype: netkeyKeygen
extdata-iv--005fs: %F2%67%45%96%41%D7%FF%10
extdata-requestversion: 8.1.0
extdata-requestortype: NETKEY_RA
extdata-keyrecord: 1
extdata-wrappeduserprivate: %94%C1%36%D3%EA%4E%36%B5%42%91%AB%47%34%C0%35%A3%6
 F%E8%10%A9%B1%25%F4%BE%9C%11%D1%B3%3D%90%AB%79
extdata-userid: jmagne
extdata-keysize: 1024
extdata-updatedby: TPS-alpha.example.com-7889
extdata-dbstatus: UPDATED
extdata-cuid: 40906145C76224192D2B
extdata-requeststatus: complete
extdata-requestid: 1
extdata-result: 1
requestType: netkeyKeygen
cn: 1
creatorsName: cn=directory manager
modifiersName: cn=directory manager
createTimestamp: 20110122021010Z
modifyTimestamp: 20110122021010Z
nsUniqueId: b2891805-1dd111b2-a6d7e85f-2c2f0000
```

Much of that information passes through the script processing unchanged, so it is entered into the new, target DRM just the same. However, some of those attributes can and should be edited, like the CN and DN being changed to match the new DRM instance. The fields which can safely be changed are listed in the configuration file for each type of key entry. (Any attribute not listed is not touched by the tool under any circumstances.)

If a field *should* be edited — meaning, the tool can update the record ID number or rename the entry — then the value is set to true in the configuration file. For example, this configuration updates the CN, DN, ID number, last modified date, and associated entry notes for all CA enrollment requests:

```
drmtool.ldif.caEnrollmentRequest.cn=true
drmtool.ldif.caEnrollmentRequest.dateOfModify=true
drmtool.ldif.caEnrollmentRequest.dn=true
drmtool.ldif.caEnrollmentRequest.extdata.keyRecord=true
drmtool.ldif.caEnrollmentRequest.extdata.requestNotes=true
drmtool.ldif.caEnrollmentRequest.requestId=true
```

If a line is set to true, then the attribute is processed in the LDIF file. By default, all possible attributes are processed. Setting a line to false means that the **`DRMTool`** skips that attribute and passes the value unchanged. For example, this leaves the last modified time unchanged so that it doesn't update for when the **`DRMTool`** runs:

```
drmtool.ldif.caEnrollmentRequest.dateOfModify=false
```

TIP

Key enrollments, records, and requests all have an optional notes attribute where administrators can enter notes about the process. When the **DRMTool** runs, it appends a note to that attribute or adds the attribute with information about the tool running, what operations were performed, and a timestamp.

```
extdata-requestnotes: [20110701150056Z]: REWRAPPED the 'existing DES3
symmetri
 c session key' with the '2048-bit RSA public key' obtained from the target s
 torage certificate + APPENDED ID offset '100000000000' + RENAMED source DRM
 naming context 'alpha.example.com-pki-kra' to target DRM naming context 'ome
 ga.example.com-pki-kra' + PROCESSED requests and key records ONLY!
```

This information is very useful for both audit and maintenance of the DRM, so it is beneficial to keep the ***extdata.requestNotes*** parameter for all of the key record types set to **true**.

IMPORTANT

Every parameter line in the default **drmtool.cfg** *must* be present in the **.cfg** file used when the tool is invoked. No line can be omitted and every line must have a valid value (true or false). If the file is not properly formatted, the **DRMTool** will fail.

The formatting of the **.cfg** file is the same as the formatting used in the instance **CS.cfg** files.

A default **.cfg** file is included with the **DRMTool** script. This file (shown in Example 26.1, “Default drmtool.cfg File”) can be copied and edited into a custom file or edited directly and used with the tool.

Example 26.1. Default drmtool.cfg File

```
drmtool.ldif.caEnrollmentRequest._000=##########################################
drmtool.ldif.caEnrollmentRequest._001=##     DRM CA Enrollment Request      ##
drmtool.ldif.caEnrollmentRequest._002=##########################################
drmtool.ldif.caEnrollmentRequest._003=##                                      ##
drmtool.ldif.caEnrollmentRequest._004=##  NEVER allow 'DRMTOOL' the ability ##
drmtool.ldif.caEnrollmentRequest._005=##  to change the CA 'naming context' ##
drmtool.ldif.caEnrollmentRequest._006=##  data in the following fields:     ##
drmtool.ldif.caEnrollmentRequest._007=##                                      ##
drmtool.ldif.caEnrollmentRequest._008=##    extdata-auth--005ftoken;uid     ##
drmtool.ldif.caEnrollmentRequest._009=##    extdata-auth--005ftoken;userid  ##
drmtool.ldif.caEnrollmentRequest._010=##    extdata-updatedby               ##
drmtool.ldif.caEnrollmentRequest._011=##                                      ##
drmtool.ldif.caEnrollmentRequest._012=##  NEVER allow 'DRMTOOL' the ability ##
drmtool.ldif.caEnrollmentRequest._013=##  to change CA 'numeric' data in    ##
drmtool.ldif.caEnrollmentRequest._014=##  the following fields:             ##
drmtool.ldif.caEnrollmentRequest._015=##                                      ##
drmtool.ldif.caEnrollmentRequest._016=##    extdata-requestId               ##
drmtool.ldif.caEnrollmentRequest._017=##                                      ##
drmtool.ldif.caEnrollmentRequest._018=##########################################
drmtool.ldif.caEnrollmentRequest.cn=true
drmtool.ldif.caEnrollmentRequest.dateOfModify=true
drmtool.ldif.caEnrollmentRequest.dn=true
drmtool.ldif.caEnrollmentRequest.extdata.keyRecord=true
drmtool.ldif.caEnrollmentRequest.extdata.requestNotes=true
drmtool.ldif.caEnrollmentRequest.requestId=true
drmtool.ldif.caKeyRecord._000=##########################################
drmtool.ldif.caKeyRecord._001=##          DRM CA Key Record           ##
drmtool.ldif.caKeyRecord._002=##########################################
drmtool.ldif.caKeyRecord._003=##                                      ##
drmtool.ldif.caKeyRecord._004=##  NEVER allow 'DRMTOOL' the ability  ##
drmtool.ldif.caKeyRecord._005=##  to change the CA 'naming context'  ##
drmtool.ldif.caKeyRecord._006=##  data in the following fields:      ##
drmtool.ldif.caKeyRecord._007=##                                      ##
drmtool.ldif.caKeyRecord._008=##    archivedBy                        ##
drmtool.ldif.caKeyRecord._009=##                                      ##
drmtool.ldif.caKeyRecord._010=##########################################
drmtool.ldif.caKeyRecord.cn=true
drmtool.ldif.caKeyRecord.dateOfModify=true
drmtool.ldif.caKeyRecord.dn=true
drmtool.ldif.caKeyRecord.privateKeyData=true
drmtool.ldif.caKeyRecord.serialno=true
drmtool.ldif.namingContext._000=################################################
drmtool.ldif.namingContext._001=##       DRM Naming Context Fields        ##
drmtool.ldif.namingContext._002=################################################
drmtool.ldif.namingContext._003=##                                            ##
drmtool.ldif.namingContext._004=##  NEVER allow 'DRMTOOL' the ability to  ##
drmtool.ldif.namingContext._005=##  change the CA 'naming context' data   ##
drmtool.ldif.namingContext._006=##  in the following 'non-KeyRecord /     ##
drmtool.ldif.namingContext._007=##  non-Request' fields (as these records ##
drmtool.ldif.namingContext._008=##  should be removed via the option to   ##
drmtool.ldif.namingContext._009=##  process requests and key records only ##
drmtool.ldif.namingContext._010=##  if this is a DRM migration):          ##
drmtool.ldif.namingContext._011=##                                          ##
drmtool.ldif.namingContext._012=##    cn                                    ##
drmtool.ldif.namingContext._013=##    sn                                    ##
drmtool.ldif.namingContext._014=##    uid                                   ##
drmtool.ldif.namingContext._015=##    uniqueMember                          ##
drmtool.ldif.namingContext._016=##                                          ##
drmtool.ldif.namingContext._017=##  NEVER allow 'DRMTOOL' the ability to  ##
drmtool.ldif.namingContext._018=##  change the DRM 'naming context' data  ##
```

```
drmtool.ldif.namingContext._019=##  in the following 'non-KeyRecord /     ##
drmtool.ldif.namingContext._020=##  non-Request' fields (as these records ##
drmtool.ldif.namingContext._021=##  should be removed via the option to   ##
drmtool.ldif.namingContext._022=##  process requests and key records only ##
drmtool.ldif.namingContext._023=##  if this is a DRM migration):          ##
drmtool.ldif.namingContext._024=##                                         ##
drmtool.ldif.namingContext._025=##      dc                                 ##
drmtool.ldif.namingContext._026=##      dn                                 ##
drmtool.ldif.namingContext._027=##      uniqueMember                       ##
drmtool.ldif.namingContext._028=##                                         ##
drmtool.ldif.namingContext._029=##  NEVER allow 'DRMTOOL' the ability to  ##
drmtool.ldif.namingContext._030=##  change the TPS 'naming context' data  ##
drmtool.ldif.namingContext._031=##  in the following 'non-KeyRecord /     ##
drmtool.ldif.namingContext._032=##  non-Request' fields (as these records ##
drmtool.ldif.namingContext._033=##  should be removed via the option to   ##
drmtool.ldif.namingContext._034=##  process requests and key records only ##
drmtool.ldif.namingContext._035=##  if this is a DRM migration):          ##
drmtool.ldif.namingContext._036=##                                         ##
drmtool.ldif.namingContext._037=##    uid                                  ##
drmtool.ldif.namingContext._038=##    uniqueMember                         ##
drmtool.ldif.namingContext._039=##                                         ##
drmtool.ldif.namingContext._040=##  If '-source_naming_context            ##
drmtool.ldif.namingContext._041=##  original source DRM naming context'   ##
drmtool.ldif.namingContext._042=##  and '-target_naming_context           ##
drmtool.ldif.namingContext._043=##  renamed target DRM naming context'    ##
drmtool.ldif.namingContext._044=##  options are specified, ALWAYS         ##
drmtool.ldif.namingContext._045=##  require 'DRMTOOL' to change the       ##
drmtool.ldif.namingContext._046=##  DRM 'naming context' data in ALL of   ##
drmtool.ldif.namingContext._047=##  the following fields in EACH of the   ##
drmtool.ldif.namingContext._048=##  following types of records:           ##
drmtool.ldif.namingContext._049=##                                         ##
drmtool.ldif.namingContext._050=##    caEnrollmentRequest:                 ##
drmtool.ldif.namingContext._051=##                                         ##
drmtool.ldif.namingContext._052=##      dn                                 ##
drmtool.ldif.namingContext._053=##      extdata-auth--005ftoken;user       ##
drmtool.ldif.namingContext._054=##      extdata-auth--005ftoken;userdn     ##
drmtool.ldif.namingContext._055=##                                         ##
drmtool.ldif.namingContext._056=##    caKeyRecord:                         ##
drmtool.ldif.namingContext._057=##                                         ##
drmtool.ldif.namingContext._058=##      dn                                 ##
drmtool.ldif.namingContext._059=##                                         ##
drmtool.ldif.namingContext._060=##    recoveryRequest:                     ##
drmtool.ldif.namingContext._061=##                                         ##
drmtool.ldif.namingContext._062=##      dn                                 ##
drmtool.ldif.namingContext._063=##                                         ##
drmtool.ldif.namingContext._064=##    tpsKeyRecord:                        ##
drmtool.ldif.namingContext._065=##                                         ##
drmtool.ldif.namingContext._066=##      dn                                 ##
drmtool.ldif.namingContext._067=##                                         ##
drmtool.ldif.namingContext._068=##    tpsNetkeyKeygenRequest:              ##
drmtool.ldif.namingContext._069=##                                         ##
drmtool.ldif.namingContext._070=##      dn                                 ##
drmtool.ldif.namingContext._071=##                                         ##
drmtool.ldif.namingContext._072=###############################################
drmtool.ldif.recoveryRequest._000=#######################################
drmtool.ldif.recoveryRequest._001=##  DRM CA / TPS Recovery Request  ##
drmtool.ldif.recoveryRequest._002=#######################################
drmtool.ldif.recoveryRequest.cn=true
drmtool.ldif.recoveryRequest.dateOfModify=true
drmtool.ldif.recoveryRequest.dn=true
```

```
drmtool.ldif.recoveryRequest.extdata.requestId=true
drmtool.ldif.recoveryRequest.extdata.requestNotes=true
drmtool.ldif.recoveryRequest.extdata.serialnumber=true
drmtool.ldif.recoveryRequest.requestId=true
drmtool.ldif.tpsKeyRecord._000=##########################################
drmtool.ldif.tpsKeyRecord._001=##         DRM TPS Key Record           ##
drmtool.ldif.tpsKeyRecord._002=##########################################
drmtool.ldif.tpsKeyRecord._003=##                                      ##
drmtool.ldif.tpsKeyRecord._004=##  NEVER allow 'DRMTOOL' the ability  ##
drmtool.ldif.tpsKeyRecord._005=##  to change the TPS 'naming context' ##
drmtool.ldif.tpsKeyRecord._006=##  data in the following fields:      ##
drmtool.ldif.tpsKeyRecord._007=##                                      ##
drmtool.ldif.tpsKeyRecord._008=##    archivedBy                        ##
drmtool.ldif.tpsKeyRecord._009=##                                      ##
drmtool.ldif.tpsKeyRecord._010=##########################################
drmtool.ldif.tpsKeyRecord.cn=true
drmtool.ldif.tpsKeyRecord.dateOfModify=true
drmtool.ldif.tpsKeyRecord.dn=true
drmtool.ldif.tpsKeyRecord.privateKeyData=true
drmtool.ldif.tpsKeyRecord.serialno=true
drmtool.ldif.tpsNetkeyKeygenRequest._000=#####################################
drmtool.ldif.tpsNetkeyKeygenRequest._001=##  DRM TPS Netkey Keygen Request  ##
drmtool.ldif.tpsNetkeyKeygenRequest._002=#####################################
drmtool.ldif.tpsNetkeyKeygenRequest._003=##                                 ##
drmtool.ldif.tpsNetkeyKeygenRequest._004=##  NEVER allow 'DRMTOOL' the      ##
drmtool.ldif.tpsNetkeyKeygenRequest._005=##  ability to change the          ##
drmtool.ldif.tpsNetkeyKeygenRequest._006=##  TPS 'naming context' data in   ##
drmtool.ldif.tpsNetkeyKeygenRequest._007=##  the following fields:          ##
drmtool.ldif.tpsNetkeyKeygenRequest._008=##                                 ##
drmtool.ldif.tpsNetkeyKeygenRequest._009=##    extdata-updatedby            ##
drmtool.ldif.tpsNetkeyKeygenRequest._010=##                                 ##
drmtool.ldif.tpsNetkeyKeygenRequest._011=#####################################
drmtool.ldif.tpsNetkeyKeygenRequest.cn=true
drmtool.ldif.tpsNetkeyKeygenRequest.dateOfModify=true
drmtool.ldif.tpsNetkeyKeygenRequest.dn=true
drmtool.ldif.tpsNetkeyKeygenRequest.extdata.keyRecord=true
drmtool.ldif.tpsNetkeyKeygenRequest.extdata.requestId=true
drmtool.ldif.tpsNetkeyKeygenRequest.extdata.requestNotes=true
drmtool.ldif.tpsNetkeyKeygenRequest.requestId=true
```

26.3. Examples

The **DRMTool** performs two operations: it can rewrap keys with a new private key, and it can renumber attributes in the LDIF file entries for key records, including enrollments and recovery requests. At least one operation (rewrap or renumber) must be performed and both can be performed in a single invocation.

Example 26.2. Rewrapping Keys

When rewrapping keys, the tool needs to be able to access the *original* NSS databases for the source DRM and its storage certificate to unwrap the keys, as well as the storage certificate for the *new* DRM, which is used to rewrap the keys.

```
DRMTool -drmtool_config_file "/usr/share/pki/java-tools/DRMTool.cfg" -
source_ldif_file "/tmp/files/originalDRM.ldif" -target_ldif_file
"/tmp/files/newDRM.ldif" -log_file "/tmp/drmtool.log" -
source_pki_security_database_path "/tmp/files/" -source_storage_token_name
"Internal Key Storage Token"  -source_storage_certificate_nickname
"storageCert cert-pki-kra"  -target_storage_certificate_file
"/tmp/files/omega.cert"
```

Example 26.3. Renumbering Keys

When multiple DRM instances are being merged into a single instance, it is important to make sure that no key or request records have conflicting CNs, DNs, serial numbers, or request ID numbers. These values can be processed to append a new, larger number to the existing values.

For the CN, the new number is the addition of the original CN plus the appended number. For example, if the CN is 4 and the append number is 1000000, the new CN is 1000004.

For serial numbers and request IDs, the value is always a digit count plus the value. So a CN of 4 has a serial number of 014, or one digit and the CN value. If the append number is 1000000, the new serial number is 071000004, for seven digits and then the sum of the append number (1000000) and the original value (4).

```
DRMTool -drmtool_config_file "/usr/share/pki/java-tools/DRMTool.cfg" -
source_ldif_file "/tmp/files/originalDRM.ldif" -target_ldif_file
"/tmp/files/newDRM.ldif" -log_file "/tmp/drmtool.log" -append_id_offset
100000000000
```

Example 26.4. Restoring the Original Numbering

If a number has been appended to key entries, as in Example 26.3, “Renumbering Keys”, that number can also be removed. Along with updating the CN, it also reconstructs any associated numbers, like serial numbers and request ID numbers. Undoing a renumbering action may be necessary if the original number wasn't large enough to prevent conflicts or as part of testing a migration or DRM consolidation process.

```
DRMTool -drmtool_config_file "/usr/share/pki/java-tools/DRMTool.cfg" -
source_ldif_file "/tmp/files/originalDRM.ldif" -target_ldif_file
"/tmp/files/newDRM.ldif" -log_file "/tmp/drmtool.log" -remove_id_offset
100000000000
```

Example 26.5. Renumbering and Rewrapping in a Single Command

Rewrapping and renumbering operations can be performed in the same invocation.

```
DRMTool -drmtool_config_file "/usr/share/pki/java-tools/DRMTool.cfg" -
source_ldif_file "/tmp/files/originalDRM.ldif" -target_ldif_file
"/tmp/files/newDRM.ldif" -log_file "/tmp/drmtool.log" -
source_pki_security_database_path "/tmp/files/" -source_storage_token_name
"Internal Key Storage Token" -source_storage_certificate_nickname "storageCert
cert-pki-kra" -target_storage_certificate_file "/tmp/files/omega.cert" -
append_id_offset 100000000000
```

26.4. Usage

This procedure rewraps the keys stored in one Certificate System 7.1 DRM and stores them in a Certificate System 8.1 DRM. This is not the only use case; the tool can be run on the same instance as both the source and target, to rewrap existing keys, or it can be used simply to copy keys from multiple DRM instances into a single instance without rewrapping the keys at all.

1. Prepare the new DRM instance and machine.
    a. Install and configure a new Red Hat Certificate System 8.1 DRM instance.

IMPORTANT

Set the storage key size and type to 2048-bit and RSA.

    b. Stop the new DRM.

```
[root@newdrm ~]# service pki-kra stop
```

    c. Create a data directory to store the exported key data from the old DRM.

```
[root@newdrm ~]# mkdir -p /export/pki
```

    d. Export the public storage certificate for the *new* DRM to a flat file in the new data directory:

```
[root@newdrm ~]# certutil -L -d /var/lib/pki-kra/alias/ -n "storageCert
cert-pki-kra" -a > /export/pki/newDRM.cert
```

    e. Stop the Directory Server instance for the new DRM, if it is on the same machine.

```
>[root@newdrm ~]# service dirsrv stop
```

    f. Export the configuration information for the new DRM.

```
[root@newdrm ~]# /usr/lib[64]/disrv/slapd-instanceName/db2ldif -n
newdrm.example.com-pki-kra -a /export/pki/newdrm.ldif
```

IMPORTANT

Be sure that the LDIF file contains a single, blank line at the end.

2. Export and prepare the key data from the old DRM instance.
   a. Create a data directory to store the exported key data.

```
[root@olddrm ~]# mkdir -p /export/pki
```

   b. Export the information from the original DRM instance using a tool like **[root@olddrm ~]# db2ldif**. This is done as part of the 7.1 to 8.1 migration steps in the DRM chapter of the Migration Guide.
   c. Copy the LDIF for the exported data into the data directory, and update the data file for change the archiving CA.

```
[root@olddrm ~]# cp /path/to/rhcs80-pki-kra.ldif /export/pki

[root@olddrm ~]# sed -i -e "s/^archivedBy:
drm_trusted_agent/archivedBy: CA/g" alpha.ldif
```

   d. Stop all of the Certificate System servers on the machine.
   e. Copy the NSS databases to the data directory. For example, for a 7.1 DRM:

```
[root@olddrm ~]# cp -p /opt/redhat-cs/alias/cert-instance-drm-cert8.db
/export/pki/cert8.db

[root@olddrm ~]# cp -p /opt/redhat-cs/alias/cert-instance-drm-key3.db
/export/pki/key3.db

[root@olddrm ~]# cp -p /opt/redhat-cs/alias/secmod.db
/export/pki/secmod.db
```

   f. Copy the **DRMTool** to the machine with the old DRM instance and pull in all its dependencies. For 7.x systems, include the **nsutil.jar** and **cmsutil.jar** files (these files are already available in 8.0 systems). For example:

```
[root@olddrm ~]# mkdir -p /usr/share/pki/java-tools

[root@olddrm ~]# mkdir -p /usr/share/java/pki

[root@olddrm ~]# cd /usr/share/java/pki

[root@olddrm ~]# sftp root@newdrm.example.com
sftp> cd /usr/share/java/pki
sftp> get nsutil.jar
sftp> get cmsutil.jar
sftp> get cstools.jar
sftp> lcd /usr/share/pki/java-tools
sftp> cd /usr/share/pki/java-tools
sftp> get DRMTool.cfg
sftp> lcd /usr/bin
sftp> cd /usr/bin
sftp> get DRMTool
sftp> quit
```

IMPORTANT

The machine must have Java 1.6.0 installed.

g. *For 7.1 DRMs.* Create a symlink from the older **`ldapjdk.jar`** file to the new 8.x location.

```
[root@olddrm ~]# ln -s /opt/redhat-cs/bin/cert/jars/ldapjdk.jar
/usr/share/java/ldapjdk.jar
```

h. Open the data directory.

```
[root@olddrm ~]#cd /export/pki
```

i. Copy the file with the public storage key from the new DRM machine to the old DRM machine. For example:

```
[root@olddrm ~]# sftp root@newdrm.example.com
sftp> cd /export/pki
sftp> get newDRM.cert
sftp> quit
```

j. If necessary, edit the default **`DRMTool.cfg`** file to use with the tool. The default file can also be used without changes.

k. Run the **`DRMTool`**; all of these parameters should be on a single line.

```
[root@olddrm ~]# DRMTool -drmtool_config_file "/usr/share/pki/java-
tools/DRMTool.cfg"
        -source_ldif_file /export/pki/rhcs80-pki-kra.ldif
        -target_ldif_file /export/pki/old2newDRM.ldif
        -log_file /export/pki/drmtool.log
        -source_pki_security_database_path /export/pki
        -source_storage_token_name 'Internal Key Storage Token'
        -source_storage_certificate_nickname 'storageCert cert-pki-kra'
        -target_storage_certificate_file /export/pki/newDRM.cert
        -append_id_offset 100000000000
        -source_drm_naming_context "olddrm.example.com-pki-kra"
        -target_drm_naming_context "newdrm.example.com-pki-kra"
        -process_requests_and_key_records_only
```

The command prompts for the password to the token stored in the original databases.

When it is done, the command creates the file specified in **`-target_ldif_file`**, **`old2newDRM.ldif`**.

l. Copy the LDIF file over to the new DRM machine. For example:

```
[root@olddrm ~]# scp /export/pki/old2newDRM.ldif
root@newdrm.example.com:/export/pki
```

IMPORTANT

Be sure that the LDIF file contains a single, blank line at the end.

3. If multiple DRM instances are being merged, then their data can be merged into a single import operation. Perform step 2 for every DRM which will be merged.

   Specify unique values for the **`-target_ldif_file`** to create separate LDIF files, and specify unique **`-append_id_offset`** values so that there are no collisions when the LDIF files are concatenated.

4. On the new DRM machine, import the LDIF file with the old key data.

   a. Open the data directory.

   ```
   [root@newdrm ~]# cd /export/pki
   ```

   b. Concatenate the new DRM configuration LDIF file and every exported LDIF for the old DRM instances. For example:

   ```
   [root@newdrm ~]# cat newdrm.ldif old2newDRM.ldif > combined.ldif
   ```

   c. Import the LDIF into the Directory Server database for the Certificate System 8.1 DRM instance.

   ```
   [root@newdrm ~]# /usr/lib[64]/disrv/slapd-instanceName/ldif2db -n
   newdrm.example.com-pki-kra -i /export/pki/combined.ldif
   ```

   d. Start the Directory Server instance for the new DRM.

   ```
   [root@newdrm ~]# service dirsrv start
   ```

   e. Start the new DRM.

```
[root@newdrm ~]# service pki-kra start
```

Index

A

ASCII to Binary tool
- example , Usage
- syntax , Syntax

AtoB tool , AtoB (Converting ASCII to Binary)

AuditVerify, AuditVerify (Audit Log Verification)

B

Binary to ASCII tool , BtoA (COnverting Binary to ASCII)
- example , Usage
- syntax , Syntax

C

CMCRequest, CMCRequest (Creating CMC Requests)

CMCResponse, CMCResponse (Parsing a CMC Response)

CMCRevoke, CMCRevoke (Signing a Revocation Request)

command-line utilities
- ASCII to Binary , AtoB (Converting ASCII to Binary)
- AuditVerify, AuditVerify (Audit Log Verification)
- Binary to ASCII , BtoA (COnverting Binary to ASCII)
- CMCRequest, CMCRequest (Creating CMC Requests)
- CMCResponse, CMCResponse (Parsing a CMC Response)
- CMCRevoke, CMCRevoke (Signing a Revocation Request)
- CRMFPopClient, CRMFPopClient (Sending an Encoded CRMF Request)
- extension joiner , ExtJoiner (Adding Cerificate Extensions to Requests)
- for adding extensions to Certificate System certificates , ExtJoiner (Adding Cerificate Extensions to Requests)
- GenIssuerAltNameExt, GenIssuerAltNameExt (Adding the Issuer Name Extention to a Request)
- PIN Generator , setpin (Generating Unique PINs for Entities)
- PKCS10Client, PKCS10Client (Generating a PKCS #10 Certificate Request)
- pkicreate, pkicreate
- pkiremove, pkiremove
- pkisilent, pkisilent
- Pretty Print Certificate , PrettyPrintCert (Printing Certificates)
- Pretty Print CRL , PrettyPrintCrl (Printing Readable CRLs))
- revoker, revoker (Sending Revocation Requests)
- sslget , sslget (Downloading Files over HTTPS)
- SubjectAltNameExt, SubjectAltNameExt (Adding the Subject Alternative Name Extension to a Request)
- TKS tool , tkstool (Managing Token Keys)

- TokenInfo , TokenInfo (Managing External Hardware Tokens)
- tpsclient , tpsclient (Debugging the TPS)

CRMFPopClient, CRMFPopClient (Sending an Encoded CRMF Request)

D

DRMTool, DRMTool (Rewrapping Private Keys)

E

Extension Joiner tool , ExtJoiner (Adding Cerificate Extensions to Requests)

extensions
- tools for generating , ExtJoiner (Adding Cerificate Extensions to Requests)

ExtJoiner tool
- example , Usage
- syntax , Syntax

G

GenIssuerAltNameExt, GenIssuerAltNameExt (Adding the Issuer Name Extention to a Request)

P

PIN Generator tool , setpin (Generating Unique PINs for Entities)
- exit codes , Exit Codes
- how it works , How setpin Works
- how PINs are stored in the directory , How PINs Are Stored in the Directory
- output file , Output File
    - checking the directory-entry status , How setpin Works
    - format , Output File
    - reasons to use an output file , How setpin Works
- overwriting existing PINs in the directory , Syntax

PKCS10Client, PKCS10Client (Generating a PKCS #10 Certificate Request)

pkicreate, pkicreate

pkiremove, pkiremove

pkisilent, pkisilent

Pretty Print Certificate tool , PrettyPrintCert (Printing Certificates)
- example , Usage
- syntax , Syntax

Pretty Print CRL tool , PrettyPrintCrl (Printing Readable CRLs))
- example , Usage

- syntax , Syntax

R

revoker, revoker (Sending Revocation Requests)

S

setpin command , The setpin Command

sslget tool , sslget (Downloading Files over HTTPS)
- syntax , Syntax

SubjectAltNameExt, SubjectAltNameExt (Adding the Subject Alternative Name Extension to a Request)

T

TKS tool
- options , Syntax
- sample , Usage
- syntax , Syntax

TokenInfo tool , TokenInfo (Managing External Hardware Tokens)
- syntax , Syntax

tpsclient tool , tpsclient (Debugging the TPS)
- syntax , Syntax